Modelos de programación 1

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Modelos de programación 1 / Gustavo Enrique Tabares Parra, /
Bogotá D.C., Fundación Universitaria del Área Andina. 2017

978-958-5459-34-2

Catalogación en la fuente Fundación Universitaria del Área Andina (Bogotá).

Edición:
Fondo editorial Areandino
Fundación Universitaria del Área Andina
Calle 71 11-14, Bogotá D.C., Colombia
Tel.: (57-1) 7 42 19 64 ext. 1228
E-mail: publicaciones@areandina.edu.co
http://www.areandina.edu.co

Primera edición: noviembre de 2017

Corrección de estilo, diagramación y edición: Dirección Nacional de Operaciones virtuales
Diseño y compilación electrónica: Dirección Nacional de Investigación

Hecho en Colombia
Made in Colombia

Modelos de programación 1

Autor: Gustavo Enrique Tabares Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

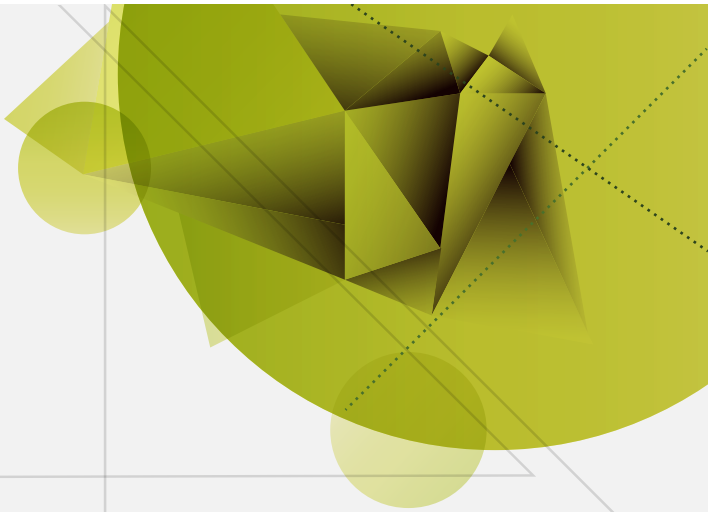
Índice

Índice

1

Unidad 1

Paradigmas de programacion



Modelos de programación I

Autor: Gustavo Enrique Tabarez Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El avance de la tecnología va de la mano con el desarrollo de software y el fundamento en los lenguajes de programación, logrando optimizar el uso del computador. La necesidad de desarrollar programas con el objetivo de resolver problemas específicos no desaparecerá.

La importancia de los lenguajes de programación, basados en una metodología novedosa, que permita la optimización de los procesos y la ejecución ideal del software, permite lograr un rendimiento y efectividad en la solución de problemas. Actualmente se encuentran gran variedad de programas y metodologías que ofrecen un procedimiento adecuado para la escritura del código y la ejecución del software.

Las computadoras procesan información usando un lenguaje de máquina, interpretando las instrucciones que reciben.Dada la complejidad de dichas instrucciones, se hace necesario el uso de un intérprete entre el computador y el programador; para tal propósito se hace uso de los lenguajes de programación.

Hoy en día existen muchos lenguajes de programación, cada uno con su propia estructura gramatical y sintaxis.

U1

Metodología

El estudiante encontrará al inicio del documento, toda la información sobre el módulo, lo que le permitirá abordar y comprender su dinámica de funcionamiento.

Como preámbulo del módulo se encontrará: el mapa conceptual, la ficha técnica, la línea de tiempo, el video de presentación, el glosario y la bibliografía.

En la primera semana se presentará una evaluación diagnóstica, cuyo fin es verificar los conceptos previos que tienen los estudiantes antes de iniciar con el tema.

Para cada unidad el estudiante contará con un documento que se conoce como guía de actividades, el cual semana a semana le informará qué actividades tiene y qué debe hacer. Asimismo, le indicará las actividades calificables en su módulo. En dicho texto se incluirán los objetivos de aprendizaje de la unidad.

La cartilla le permitirá abordar los temas desde la introducción hasta conocer los paradigmas y el concepto de la programación. Se recomienda leer cuidadosamente y en forma secuencial la información para tener una compresión adecuada de los temas.

La actividad de actividad evaluativa - evaluación diagnóstica, le permitirá al estudiante evaluar su nivel de conocimiento referente al tema de modelos de programación, y profundizar en aquellos contenidos que se encuentre más débil.

Se recomienda tener muy presente las lecturas complementarias de la semana donde se puede profundizar en el tema.

U1

Desarrollo temático

Paradigmas de programación

La evolución tecnológica que sufre el software, está sometido a cambios constantes cada vez más exigentes.Desde sus comienzos el código de máquina y la programación en lenguaje ensamblador, han tenido una transformación trascendental, pasando por varios momentos, dando la importancia siempre a la forma de pensar en la solución de problemas e implementando alguno de los siguientes paradigmas:

- Programación modular.
- Programación procedural.
- Programación estructurada.
- Programación imperativa.
- Programación declarativa.
- Programación funcional.
- Programación Orientada a Objetos.

Programación modular

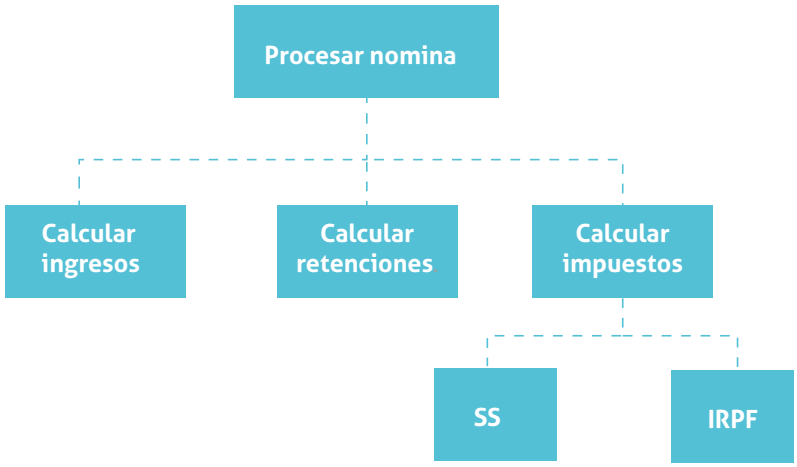


Figura 1. Programación modular.
Fuente: http://www.iesjuandelacierva.com/paginade/fupro/apuntes/ut10.htm

Se compone de bloques separados llamados módulos, los cuales están de forma independientes, se pueden relacionar por medio de interfaces que estén bien definidas.

Programación procedural

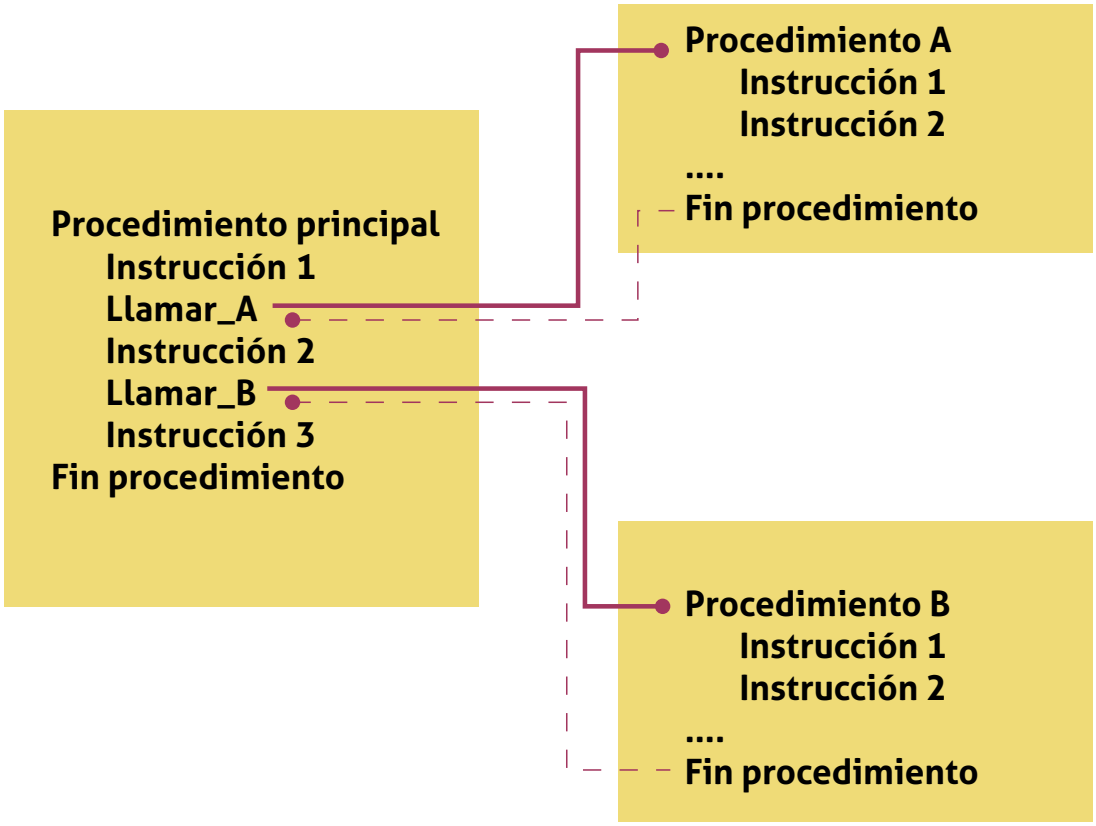


Figura 2. Programación procedural ejecución del código.
Fuente: http://apuntesdesistemasinformatica.blogspot.com/2012/09/programacion-procedural-i.html

Se compone de procedimientos formando un solo programa donde se encuentran subrutinas, métodos o funciones, los cuales son fragmentos de código que pueden ser llamados desde diferentes partes en el momento de la ejecución de un programa, incluyendo otros procedimientos. Como ejemplo de este paradigma está el lenguaje de programación ALGOL.

Programación estructurada

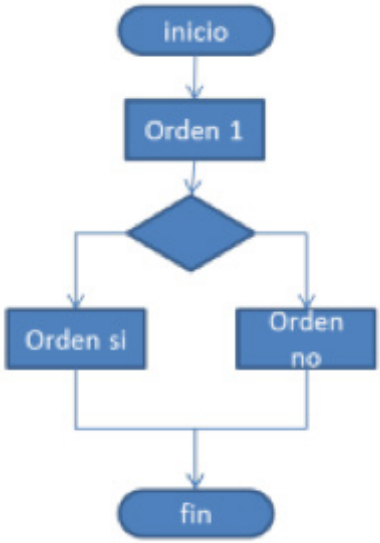


Figura 3. Programación Estructurada.
Fuente: http://sistemasumma.com/2012/09/18/primeros-pasos-en-programacion-vi-programacion-estructurada/

Se compone del uso de tres estructuras de control, la secuencial, condicional e iterativa. Los desarrollos están compuestos de partes menores como punto de entrada, y se trata de aislarlos para evitar la complejidad que introducen los efectos colaterales.Como ejemplo de esta programación están los lenguajes PASCAL, C, ADA.

Programación imperativa

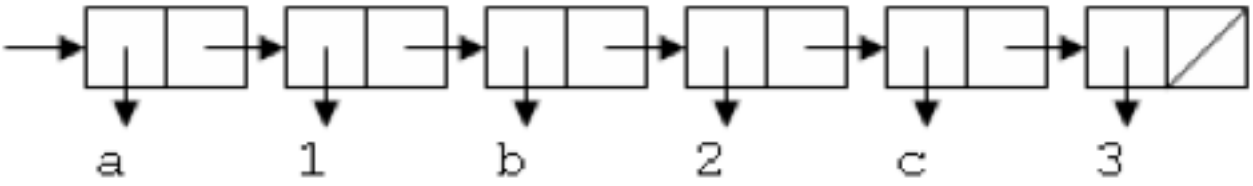


Figura 4. Programación Imperativa.
Fuente: Propia.

Se compone por programas en términos de estados y de instrucciones (líneas de código) secuenciales que modifican dichos estados; como ejemplo de esta programación están los lenguajes C, BASIC.

Programación declarativa

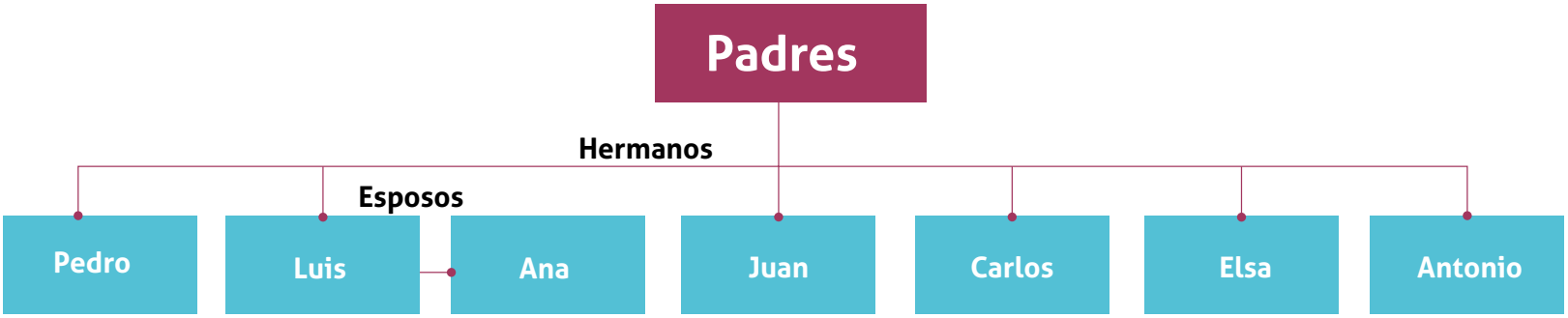


Figura 5. Programación Declarativa.
Fuente: Propia.

Se compone a través de condiciones, proposiciones o restricciones, de donde se obtiene la solución mediante la implementación de reglas de control.Como ejemplo de esta programación está el lenguaje PROLOG.

Programación funcional

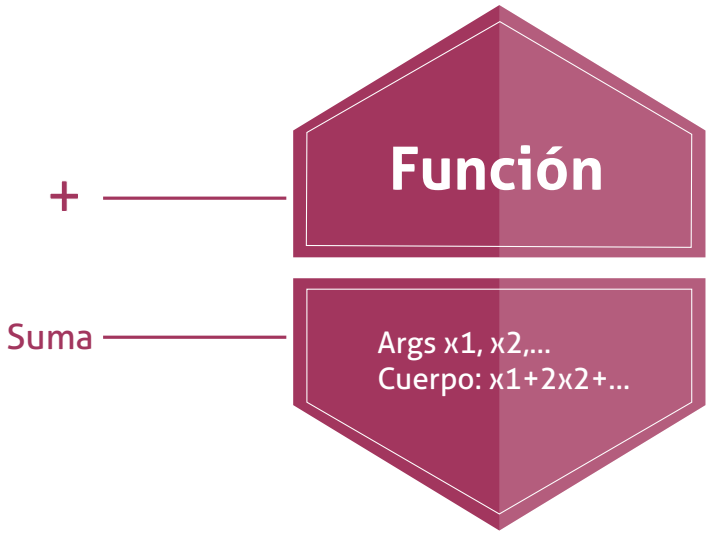


Figura 6. Programación Funcional.
Fuente: Propia.

Se compone de una secuencia de aplicación de funciones, sin tener en cuenta el concepto de estado del cómputo y sin precisar el uso de las estructuras de programación estructurada; como ejemplo de esta programación está el lenguaje LISP, HASKELL.

Programación orientada a objetos

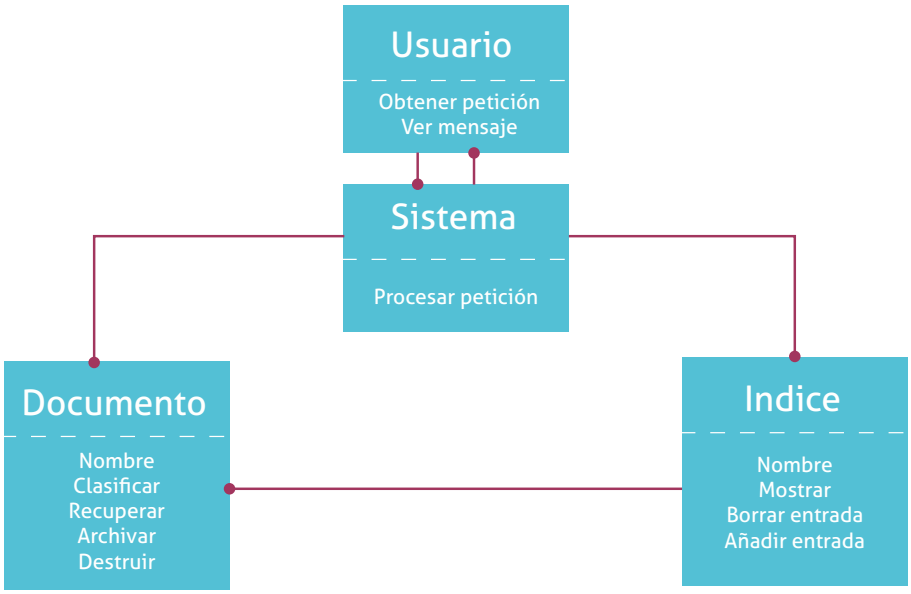


Figura 7. Programación Orientada a Objetos.
Fuente: http://veneinfor-oop.blogspot.com/

Es un paradigma que consiste en la abstracción del mundo real mediante el uso de objetos. Dichos objetos poseen propiedades (atributos) que dan su definición y le permiten realizar operaciones e interactuar con otros objetos mediante el intercambio de mensajes. Es el resultado evolutivo de los paradigmas de la programación Procedural, Estructurada y Modular. Como ejemplo de esta programación están los lenguajes JAVA, SMALLTALK, PYTHON, C++.

En el proceso de un proyecto de informática para el desarrollo de software, existen una serie de etapas ordenadas donde la programación hace parte de una de ellas.Existen varios tipos de desarrollo que definen esas etapas. Una de las metodologías de desarrollo más usadas es la en cascada, que posee a grandes rasgo las siguientes fases:

Análisis: es la etapa más importante, da inicio al proyecto y analiza el problema a solucionar en su totalidad. Se interpreta el problema para lograr una solución adecuada y eficaz del mismo.

Diseño: después del análisis anterior, viene el diseño de la solución del problema, donde se define la estructura, los módulos, algoritmos, etc.; que van a complementar la solución.Estas dos etapas son la mayor parte de la ejecución del proyecto.

Implementación: partiendo de la etapa anterior viene la etapa de implementación que se traduce a un lenguaje de programación y se desarrolla el código que conforma el programa solución.

Pruebas: son las evaluaciones y mediciones a las que se lleva el programa para terminar de ajustar el software.

Producción: puesta en producción del programa totalmente ajustado.

Mantenimiento: son las adaptaciones que se llevan a cabo para su correcto funcionamiento operativo y tecnológico.

Evolución de los modelos y lenguajes de programación

La evolución de los lenguajes de programación se analizará desde el nivel de abstracción operacional y con respecto al nivel de abstracción de datos.

La abstracción operacional se define como la agrupación del código en operaciones a modo de procedimientos o funciones,con el fin de que este sea más legible y de mejor desempeño. La abstracción de datos es la capacidad de identificar los datos independientemente de la forma en que se implementen.

La figura No 8, muestra de forma resumida cómo los lenguajes de programación han tenido una evolución altamente considerable a través de los tiempos, donde el lenguaje es más compacto y eficiente. Se presenta desde sus inicios con el lenguaje ensamblador de un nivel muy bajo de abstracción y llega hasta los lenguajes de alto nivel de abstracción y los orientados a objetos. En el centro de la figura se muestran los lenguajes más representativos en ambos niveles de abstracción; al lado izquierdo están las características novedosas de dichos lenguajes en la abstracción funcional y a la derecha se muestran los detalles más relativos a la abstracción de datos.

Figura 8. Nivel de abstracción de los lenguajes.
Fuente: Propia.

Lenguajes de programación de alto nivel

Los lenguajes de alto nivel facilitan a los programadores poder escribir programas mediante el uso de expresiones sintácticas del idioma inglés, como: prinft, while, for, public, if, case, etc.; y lograr construir líneas de código en un programa:

```
if( valor > 0)

        printf("El valor numérico es positivo");
```

Si interpretamos la instrucción se valida una condición que puede ser verdadera o falsa; si la condición evaluada es verdadera se ejecuta la siguiente instrucción que indica que se debe imprimir unmensaje por pantalla: El valor numérico es positivo. En caso contrario no pasa nada.

Se les considera lenguajes de alto nivel a la forma de poder interactuar con la computadora en un lenguaje normal y comprensivo para los usuarios. Entre los lenguajes de alto nivel están:Basic, Fortran, Cobol, Java, C.

Los lenguajes de bajo nivel son aquellos que tienen una comunicación más directa con la máquina, bajo un lenguaje de máquina.

Evolución de los lenguajes de programación

Antes de mencionar los lenguajes de programación es importante saber el modo de comunicación y las ordenes que recibe el computador en un código binario totalmente legible para él, ya que es el único lenguaje entendible para su unidad de procesamiento.Esta labor es llevada a cabo por un intérprete o compilador.

El intérprete traduce las instrucciones del lenguaje y favorece la interactividad, la depuración y ejecución del programa; entre los lenguajes interpretes se tienen: BASIC, LOGO, etc.

A continuación se analizan algunos lenguajes que surgieron para facilitar la comunicación entre los usuarios y las computadoras:

FORTRAN: Se dio al comienzo de la década de los años 50, el informático y programador Jhon Backus trabajaba con SSEC (SelectiveSequenceElectronicCalculator), en 1956 se terminó de crear el compilador (FORmulaTRANslator), el cual estaba destinado a la resolución de problemas científicos-técnicos fácilmente de aprender.

COBOL (Common Business OrientedLanguage): en los años 50 se creó un lenguaje orientado a funciones administrativas, su evolución fue muy rápida logrando la creación de nuevas actualizaciones, en la actualidad se usa en entidades bancarias.

BASIC: creado por los profesores Jhon G. Kemeny y Thomas E. Kurt en 1964, crearon un lenguaje para facilitar la comunicación compartida entre los estudiantes, fácil de aprender y de gran efectividad en las tareas científicas. Tuvo una gran aceptación por su fácil manejo y poco consumo de memoria pero no era un lenguaje muy potente.

C: Los creadores del sistema operativo UNIX en 1969, Kenneth Thompson y Dennis Ritchie trabajando en los laboratorios BELL se dio la creación del lenguaje de programación C.,siendo uno de los lenguajes más portables del mercado y de gran uso en los programas actuales con C++ enfocado a la programación orientada a objetos.

JAVA: Es un lenguaje de programación completamente orientado a objetos, los creadores de lenguaje son Sun Microsystems en los años 90. Hoy en día se encuentra el lenguaje de programación en múltiples equipos con la tecnología Java en redes, Internet, grandes ordenadores, equipos móviles, portátiles, juegos, simuladores, cards, etc.

Los objetos

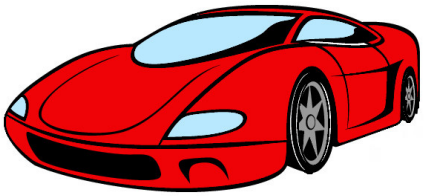


Imagen 1.
Fuente: com-hd.net

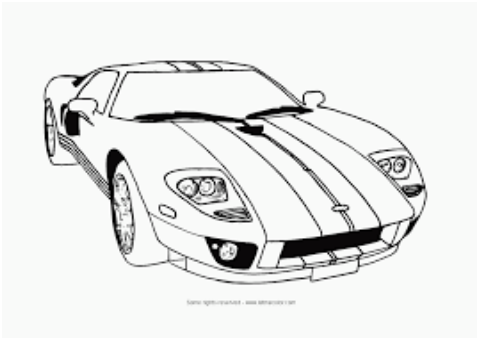


Imagen 2.
Fuente: www.pintar-colorear-dibujos.com

Imagen 3.
Fuente: www.pintar-colorear-dibujos.com

Modelo.

Marca.

Color.

Velocidad.

Acelerar.

Desacelerar.

Apagar.

Arrancar.

Los objetos en programación consisten en dividir en clases los objetos. Partiendo del concepto objeto se puede observar en el mundo actual variedad de objetos que tienen rasgos comunes o pueden ser completamente diferentes. Se pueden agrupar o relacionar de acuerdo a sus características y comportamientos del mundo real, como: tamaño, color, forma, etc.

Características de los objetos.

Atributos:son las características propias de un objeto, estos se definen a través de variables

que almacenan la información.

Métodos: son procedimientos o funciones que determinan las acciones que pueden realizar los objetos, ya sea sobre sí mismos o sobre otros objetos.

Abstracción

El nivel de abstracción está relacionado con la capacidad que tiene el ser humano para identificar los atributos de un objeto y abstraer del problema las variables que intervienen para ser usadas en la construcción del algoritmo.

Cuando se habla de la programación orientada a objetos no es una mejora de programación a los lenguajes existentes, sino un cambio a la forma de pensar en la abstracción para la solución de un problema.

La programación orientada a objetos es un paradigma que abstrae las propiedades de un objeto adaptándolas a un software.

Analicemos un ejemplo

Cuando nos detenemos a pensar cómo abordar un problema, debemos reflexionar sobre los objetos que interactúan en él, de forma que podamos comprender sus relaciones y la integridad en el planteamiento del mismo.

Identificar los atributos y métodos para definir sus comportamientos, es el objetivo del siguiente ejemplo, donde se plantea una situación en la que se debe realizar una abstracción cuando se da una solución pensada en objetos.

Imaginemos la siguiente situación: un domingo por la tarde estoy en casa viendo la televisión, y de repente mi madre siente un fuerte dolor de cabeza; como es natural, lo primero que hago es tratar de encontrar una caja de aspirinas.

Lo que acabo de describir es una situación que probablemente no resulte muy extraña a muchos de nosotros. Vamos a verla en clave de objetos: el objeto hijo ha recibido un mensaje procedente del objeto madre. El objeto hijo responde al mensaje o evento ocurrido mediante una acción: buscar aspirinas. La madre no tiene que decirle al hijo dónde debe buscar, es responsabilidad del hijo resolver el problema como considere más oportuno. Al objeto madre le basta con haber emitido un mensaje. Continuemos con la historia.

El hijo no encuentra aspirinas en el botiquín y decide acudir a la farmacia de guardia más cercana para comprar aspirinas. En la farmacia es atendido por una señorita que le pregunta qué desea, a lo que el hijo responde: "una caja de aspirinas, por favor". La farmacéutica desaparece para regresar al poco tiempo con una caja de aspirinas en la mano. El hijo paga el importe, se despide y vuelve a su casa. Allí le da un comprimido a su madre, la cual al cabo de un rato comienza a experimentar una notable mejoría hasta la completa desaparición del dolor de cabeza.

El hijo, como objeto responsable de un cometido, sabe lo que debe hacer hasta conseguir una aspirina. Para ello entra en relación con un nuevo objeto, la farmacéutica, quien responde al mensaje o evento de petición del objeto hijo con la búsqueda de la aspirina. El objeto farmacéutica es ahora el responsable de la búsqueda de la aspirina. El objeto farmacéutica lanza un mensaje al objeto hijo solicitando el pago del impor-

te, y el objeto hijo responde a tal evento con la acción de pagar.

Como hemos podido ver, en esta situación nos hemos encontrado con objetos que se diferenciaban de los demás por un conjunto de características o propiedades, y por un conjunto de acciones que realizaban en respuesta a unos eventos que se originaban en otros objetos o en el entorno.

También podemos darnos cuenta de que, aunque todos los objetos tienen propiedades distintas, como el color del cabello, el grado de simpatía o el peso, todos tienen un conjunto de atributos en común por ser ejemplos de una entidad superior llamada "ser humano". A este patrón de objetos (en nuestro caso "ser humano") lo llamaremos clase. (Izquierdo, 2007).

Programación basada en componentes

"La programación orientada en componentes surge como alternativa de la programación orientada a objetos de los sistemas abiertos, debido a que la programación orientada a objetos no precisa la diferencia entre aspectos computacionales y los composicionales de las aplicaciones, esta programación nace como necesidad global de componentes de software, para la reutilización componentes de software y poder crear aplicaciones mucho más rápidas". (Lidia Fuentes, 2009)

Programación orientada a aspectos

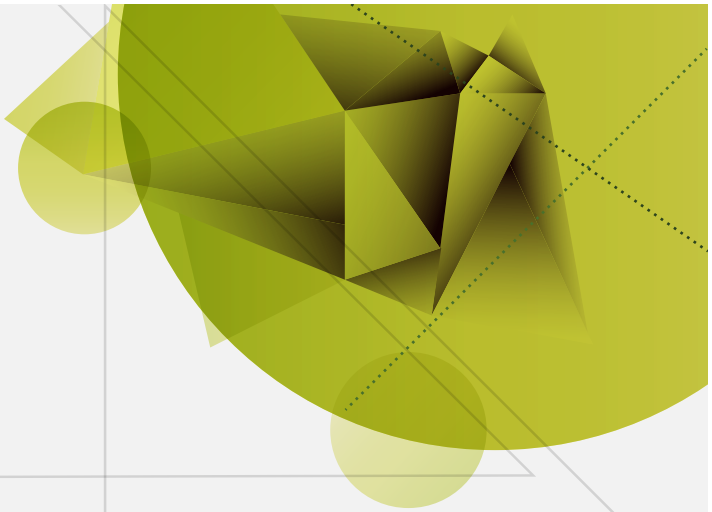
La programación orientada a aspectos complementa a la programación orientada a objetos facilitando al programador modificar de forma dinámica el modelo orientado a objetos, creando un sistema que se adapta fácilmente a las especificaciones definidas.

Cuando se crea una aplicación web donde el usuario ingresa datos a través de un formulario y este da una respuesta sencilla, pues se trata de un proceso simple, pero cuando la aplicación crece y se requiere cubrir otros requisitos secundarios, la aplicación no se hace necesario tener en cuenta estos aspectos pues su función principal es recibir información a través de una entrada y procesarla. La programación orientada a aspectos permite modificar el modelo estático incluyendo nuevo código que implemente los requerimientos necesarios sin modificar el modelo fuente original. (Moreno, 2014).

1

Unidad 1

Programación estructurada



Modelos de programación I

Autor:Gustavo Enrique Tabarez Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El avance de la tecnología va de la mano con el desarrollo de software y el fundamento en los lenguajes de programación, logrando optimizar el uso del computador. La necesidad de desarrollar programas con el objetivo de resolver problemas específicos no desaparecerá.

La importancia de los lenguajes de programación, basados en una metodología novedosa, que permita la optimización de los procesos y la ejecución ideal del software, permite lograr un rendimiento y efectividad en la solución de problemas.Actualmente se encuentran gran variedad de programas y metodologías que ofrecen un procedimiento adecuado para la escritura del código y la ejecución del software.

U1

Metodología

Para cada unidad el estudiante cuenta con un documento que le describirá las actividades a realizar para la semana; dicho documento se denomina *Guía de actividades*. Asimismo le informa las actividades calificables e incluye los objetivos de aprendizaje. Antes de iniciar usted debe leerlo muy atentamente.

La cartilla le permitirá abordar los temas a tratar en cada semana. Se recomienda leer cuidadosamente y en forma secuencial la información para tener una compresión adecuada de ellos.

De otro lado, cada semana cuenta con lecturas complementarias, recursos para el aprendizaje, actividades de repaso y videoconferencias que le permitirán aclarar dudas y/o profundizar en la temática semanal.

La actividad evaluativa, le permitirá al estudiante, evaluar su nivel de conocimiento referente al tema y poder profundizar en aquellos temas que se encuentre más débil.

U1

Desarrollo temático

Programación estructurada

La programación estructurada se caracteriza por ser de fácil compresión en la construcción de programas, permite la corrección de estos después de su creación. La interpretación del código que conforma el programa es mucho más sencilla de entender.

Está basada en una metodología de desarrollo que permite la construcción de programación mediante procesos sucesivos, se define un proceso global y se divide en módulos más pequeños que permiten la compresión y solución del problema, mediante la integración de los módulos por parte de los programadores, cuando se presentan problemas de codificación estos son fáciles de detectar y corregir.

La representación gráfica que se realiza mediante diagramas de flujo, permiten representar la forma lógica que tendrá la solución planteada mostrando las posibles entradas que tendrá el programa, los procesos que se dan, y las salidas que genera la solución.

La programación estructurada está basada en la implementación de estructuras simples conocidas como secuencias de selección e interacción; que hacen parte de los lenguajes imperativos de sentencias, haciendo uso de esquemas de programación sencillos que permiten construir sistemas muy complejos y de fácil entendimiento.

La programación lleva un orden claro, lógico, disciplinado que permite escribir programas y modificarlos fácilmente.

Para dar solución a un determinado problema a través de la programación estructurada, se debe aplicar las funciones que debe cumplir en general el programa, para ir dividiéndolo en subfunciones más pequeñas, para ser programados y construir nuevos módulos para ser insertados en forma correcta donde correspondan.

Un programa cumple con un diseño estructurado si contiene las siguientes condiciones:

- Está basado en el teorema de estructura.
- Si está debidamente documentado.

La programación estructurada está basada el teorema de Edsge rDijkstra, donde dice que todo programa se puede escribir partiendo de tres estructuras básicas de control:

Secuencia: comprende un bloque o agrupación secuencial de instrucciones que se ejecutan de forma sucesiva, una seguida de la otra.

Selección: Comprende los condicionales, simple, dobles, ejemplo: if (condición) then

instrucción1,else instrucción2.

Iteración: Los ciclos o bucles condicionales “for”, “while” y “do while”, que ejecutan las instrucciones el ciclos repetitivos mientras la condición se cumpla.

Ventajas y desventajas de la programación estructurada

Ventajas:

Los programas son más fáciles de leer.

La estructura de los programas es clara.

Reducción de esfuerzo en las pruebas.

Se incrementa el rendimiento del programador.

Desventajas:

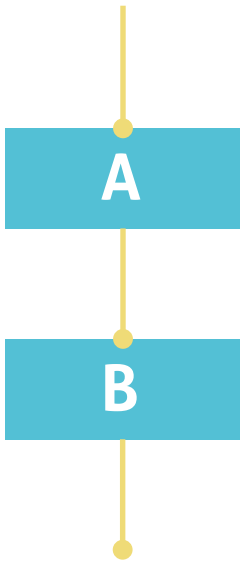
Se obtiene un único bloque del programa que puede resultar ser muy grande y puede ser problemático y complejo en la interpretación del código fuente.

Sentencias de la programación estructurada

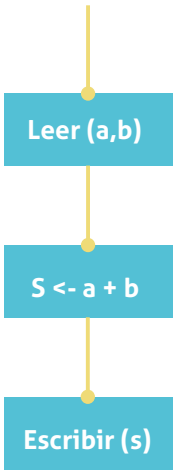
Estructura secuencial

Las instrucciones se ejecutan una a una en orden secuencial, llevando el orden con el que aparece en el programa.

Gráficamente es un proceso seguido de otro, con una sola entrada y una única salida.



En la figura se observa una secuencia del proceso A al proceso B, llevando un orden en la definición de los bloques del programa, ejecutando una instrucción o programa completamente.

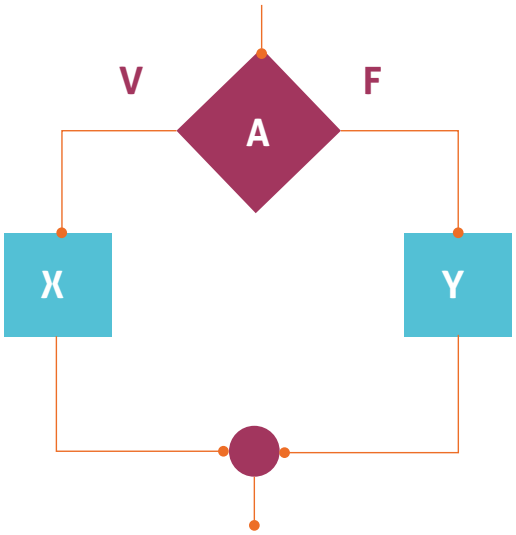


En el ejemplo es necesario los valores iniciales de a,b para luego realizar el cálculo correspondiente y dar como salida el resultado.

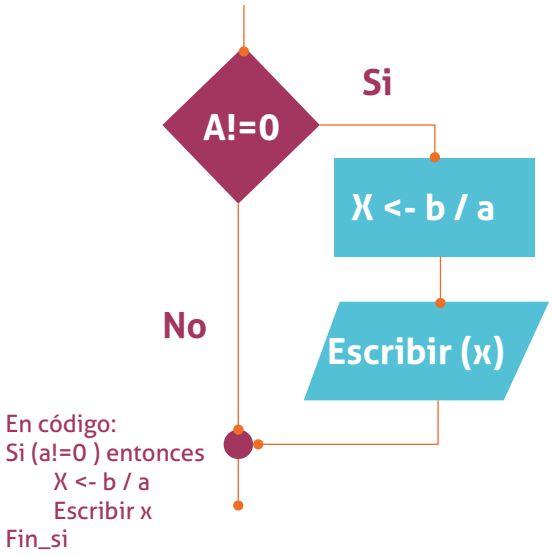
Estructura selectiva

Esta estructura está basada en la evaluación de una condición verdadera y falsa, planteando una solución posible entre dos alternativas de acuerdo al resultado de una evaluación. En los lenguajes de programación es conocido con la instrucción IF.

Ejemplo:



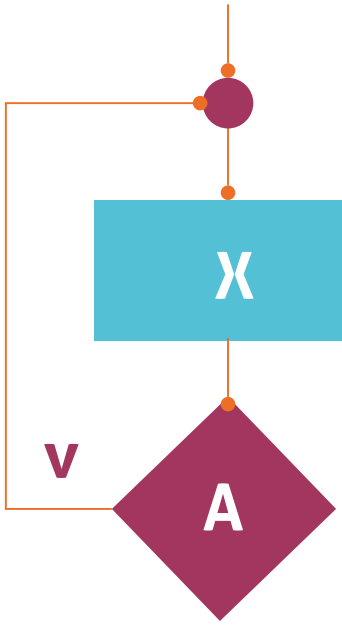
En la figura se representa un diagrama con la estructura selectiva, donde la variable A representa la condición que se evalúa, tiene una sola entrada, el resultado que genere la condición verdadera o falsa solo se ejecuta una operación. Si se ejecuta la operación X es porque la condición que se evaluó arrojo verdadero, de lo contrario es porque la condición que se evaluó arrojo falso. Dentro de la operación X, se puede ejecutar múltiples operaciones.



Estructura repetitiva

Esta estructura se basa en la ejecución de ciclos repetitivos (iterativos) conocidos como: Hacer, Mientras, Ejecute mientras. Consisten en ejecutar repetidas instrucciones mientras se esté cumpliendo la condición.

Ejemplo:

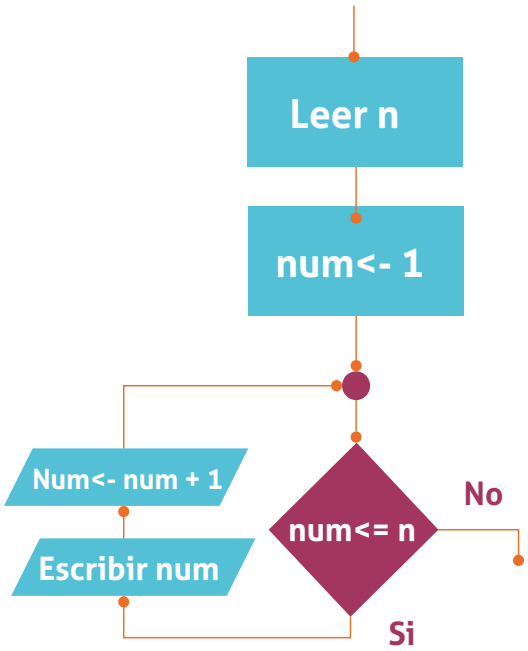


La estructura repite el proceso X mientras la condición sea verdadera. El proceso X tiene una sola entrada, puede contener múltiples instrucciones y genera una salida.

Veamos un ejemplo aplicando la estructura repetitiva "mientras".

Escribir los números enteros de 1 a N.

El diagrama de flujo para una solución estructurada es el siguiente:



La solución en programación estructurada es:

Leer n

num<-1

mientrasnum<= n hacer

Escribir num

num<- + 1

fin_mientras

Haciendo unaprueba de escritorio observamos que el diagrama realiza los siguientes pasos:

1. Lee el valor de n.
2. La estructura repetitiva "mientras" evalúa la condición inicial y si ésta es verdadera, entra al ciclo.

3. Escribe el valor de num y suma 1 a la variable num.
4. El ciclo "mientras" evalúa nuevamente la condición y si se cumple entra nuevamente al ciclo.
5. Se repite los pasos 3 y 4.

El ciclo termina hasta que la condición arroje falso.

Aplicaciones de la programación estructurada

La programación estructurada, la forma de construir programas, requiere creatividad, habilidad y destreza por parte del programador. Algunas ventajas que podemos lograr a través de esta programación es:

a. El código que forma el programa es más fácil de entender, este puede ser interpretado de forma secuencial, de línea a línea sin realizar salto entre ellas, hasta recorrer el código y entender su lógica.
b. Cada instrucción está muy relacionada o dependiente de la anterior durante todo el código de la aplicación.
c. Los errores se pueden detectar y corregir de forma fácil ya que estos se logran detectar por la estructura y son más visibles.
d. El proceso de mantenimiento del software de programas estructurados es más fácil y por ende, tienen un menor costo.
e. Los programas son sencillos y rápidos por su estructura.
f. La documentación es sencilla y clara debido a que el programa está dividido en bloques.
g. Los programas que están bajo este paradigma, presentan una estructura sencilla, una organización clara y una presentación del código altamente legible.

Ejemplo de aplicación de la Programación Estructurada

Escribir un procedimiento para calcular el área de cualquier triángulo rectángulo;en él se debe pedir al usuario que ingrese los valores de la altura y la base del triángulo.

Análisis del problema

Formular el problema: Ya está claramente planteado.

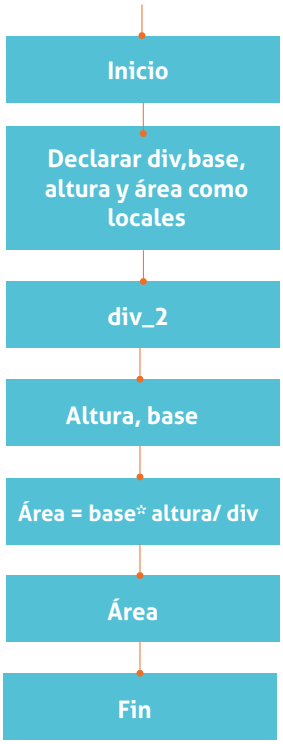
Resultados esperados: Un procedimiento que permita calcular el área de cualquier triángulo rectángulo.

Datos disponibles: base y altura del triángulo (se deben solicitar al usuario). El estudiante debe preguntarse si sus conocimientos actuales de matemáticas le permiten resolver este problema; de no ser así, debe plantear una estrategia para obtener los conocimientos requeridos.

Restricciones: Los valores de base y altura son variables y se deben solicitar al usuario.

Procesos necesarios: definir variables; asignar el valor 2 a la constante div; solicitar al usuario el valor de la altura del triángulo; solicitar al usuario el valor de la base; aplicar la fórmula de área; mostrar el resultado.

Diseño del algoritmo



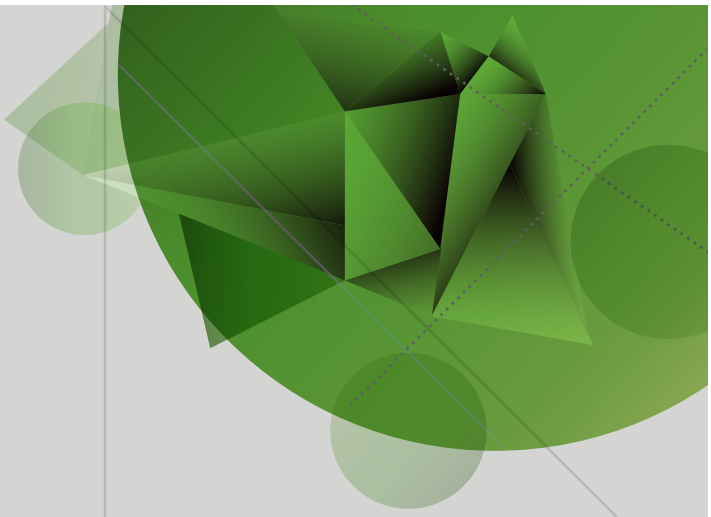
En este ejemplo, el procedimiento triángulo rectángulo también está compuesto únicamente por una estructura secuencial de instrucciones. En ella se utilizan las primitivas "pregunta" y "respuesta" para permitir que el usuario del programa suministre al programa los datos "altura" y "base" del triángulo. De esta forma, se logra un procedimiento generalizado para calcular el área de cualquier triángulo rectángulo. En otras palabras, cada vez que se ejecute el procedimiento triángulo rectángulo, este le preguntará al usuario cuál es la altura y la base del triángulo del cual desea calcular su área.

Tanto en la utilización de la estructura secuencial, como en las dos que se verán más adelante, es muy importante que los estudiantes reflexionen y determinen el orden de ejecución de las instrucciones (posición), ya que la conmutatividad no es una propiedad aplicable a los algoritmos. El lenguaje algorítmico, al igual que el lenguaje formal de las matemáticas, tiene carácter gráfico y posicional; busca la precisión, el rigor, la abreviación y la universalidad; y, su finalidad fundamental consiste en obtener resultados internamente consistentes (Onrubia&Rochera&Barbarà, 2001).

2

Unidad 2

Programación orientada a objetos



Modelos de programación I

Autor: Gustavo Enrique Tabarez Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

La exigencia en el avance tecnológico hace que hoy en día se desarrollen aplicaciones informáticas mucho más novedosas, y exigen que el desarrollo de software cumpla con los estándares y métricas de calidad para que tenga un mayor impacto la comunidad y se mantenga por un periodo de tiempo razonable.

La programación orientada a objetos, se ha posicionado en el mundo del desarrollo de software, y ha sido el paradigma que actualmente se implementa en la mayoría de las aplicaciones en el mundo. No es un paradigma novedoso, sus inicios se dieron en los años 60, con el lenguaje de programación Simula. Se basa en la creación de modelos que son extraídos de situaciones problemáticas del mundo real y que se implementan en soluciones de desarrollo de software basado en el modelo.

Actualmente nos encontramos rodeados de objetos que pueden hacer parte de una solución informática por sus atributos y comportamientos. Como objetos podemos tener el nombre de una persona, un empleado, una ventana, un botón, un marcador, etc.

Los atributos permiten identificar las características de los objetos; los comportamientos son las acciones que manifiestan los objetos dentro de una aplicación informática.

En esta unidad, veremos cómo podemos extraer de una situación problemática los objetos que interactúan e identificaremos sus comportamientos para dar solución mediante una aplicación tecnológica.

U2

Metodología

Para cada unidad el estudiante cuenta con un documento texto, el cual le describe las actividades a realizar para la semana; dicho documento se denomina Guía de actividades. Asimismo le informa las actividades calificables e incluye los objetivos de aprendizaje. Antes de iniciar usted debe leerlo muy atentamente.

La cartilla le permitirá abordar los temas a tratar en cada semana. Se recomienda leer cuidadosamente y en forma secuencial la información para tener una compresión adecuada de ellos.

De otro lado, cada semana cuenta con lecturas complementarias, recursos para el aprendizaje, actividades de repaso y videoconferencias que le permitirán aclarar dudas y/o profundizar en la temática semanal.

La actividad evaluativa, le permitirá al estudiante, evaluar su nivel de conocimiento referente al tema y poder profundizar en aquellos temas que se encuentre más débil.

U2

Desarrollo temático

Programación orientada a objetos

Los objetos puede tomar diferentes comportamientos de acuerdo a las necesidades que se tengan, con la POO se puede reutilizar estos comportamientos y ocultar los que no se usen. Para comprender bien la programación orientada a objetos estudiaremos los conceptos de clase, objeto herencia, encapsulación y polimorfismo. Teniendo claro estos conceptos podemos comprender la estructura funcional de la POO, desde sus clases y jerarquías..

Clase: es una abstracción general que se realiza de una situación que requiera un desarrollo informático. Dentro de un entorno empresarial existen empleados, clientes, proveedores; que poseen características comunes entre ellos, como son (documento, nombre, dirección, teléfono, género, etc.), que al realizar una abstracción podríamos definir la clase persona como el elemento que los agrupa con las mismas características, y cada uno de ellos (empleados, clientes, proveedores) los objetos que se derivan de la clase persona.

Veamos un ejemplo más claro, un empleado tiene las características (documento, nombre, dirección, teléfono, género) que lo identifica y podríamos crear una clase **empleado.**

Un cliente tiene las mimas características de (documento, nombre, dirección, teléfono, género) que lo identifica y podríamos también crear una clase cliente.

Un proveedor también tiene un (documento, nombre, dirección, teléfono, género) que igual que al empleado y el cliente también lo identifica y podríamos crear otra clase **proveedor.**

Al final tendríamos tres clases empleado, cliente, proveedor con las mismas características dentro una aplicación y estaríamos aplicando POO, pero redundando información en el uso de sus características. La clase persona agrupa las características, atributos generales de los tres y los agrupa en una clase, permitiendo que los objetos empleados, clientes y proveedores puedan acceder a estas características cuando lo requieran. En la gráfica No. 1 se detalla con más claridad esta situación.

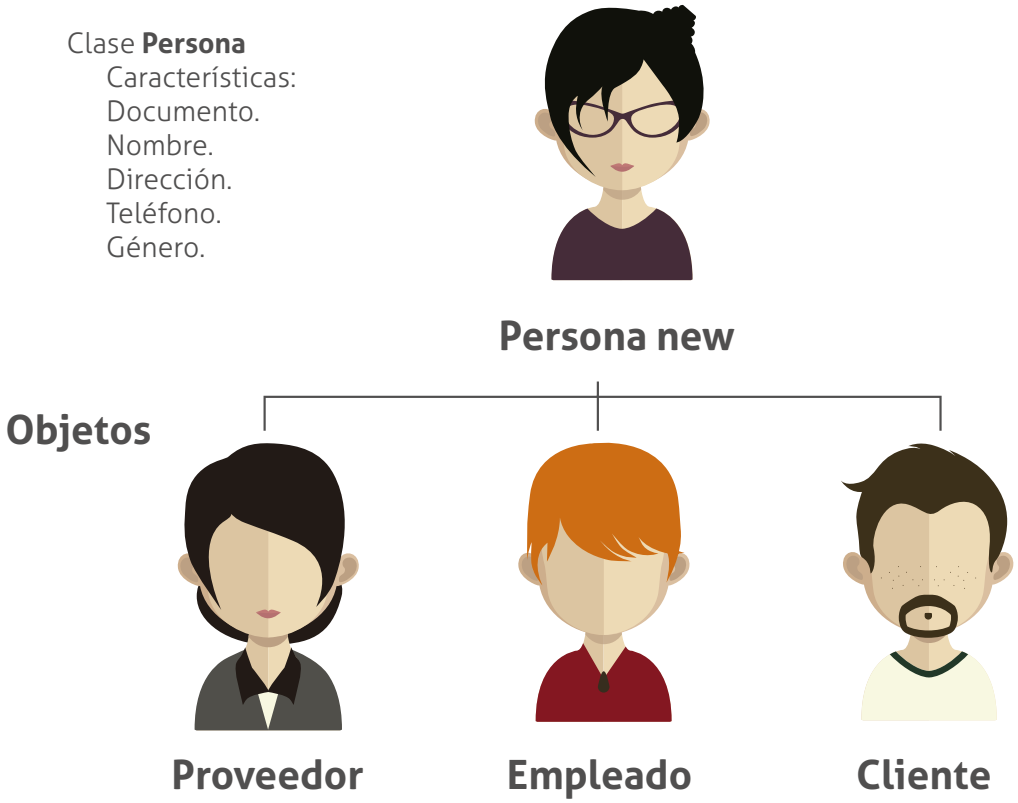


Figura 1. Clase persona, instanciando tres objetos.
Fuente: Propia.

Otro ejemplo

Dentro de una empresa que se dedica a la fabricación de varios automóviles de diferentes modelos existen automóviles personales, de carreras, deportivos, clásicos; estos poseen características comunes entre ellos, como son (números de motor, color, tamaño, modelo, número de ruedas, etc.), al realizar una abstracción podríamos definir la clase **auto** como el elemento que los agrupa con las mismas características, y cada uno de los automóviles (personales, de carreras, deportivos, clásicos) serían los objetos que se derivan de la clase **auto.**

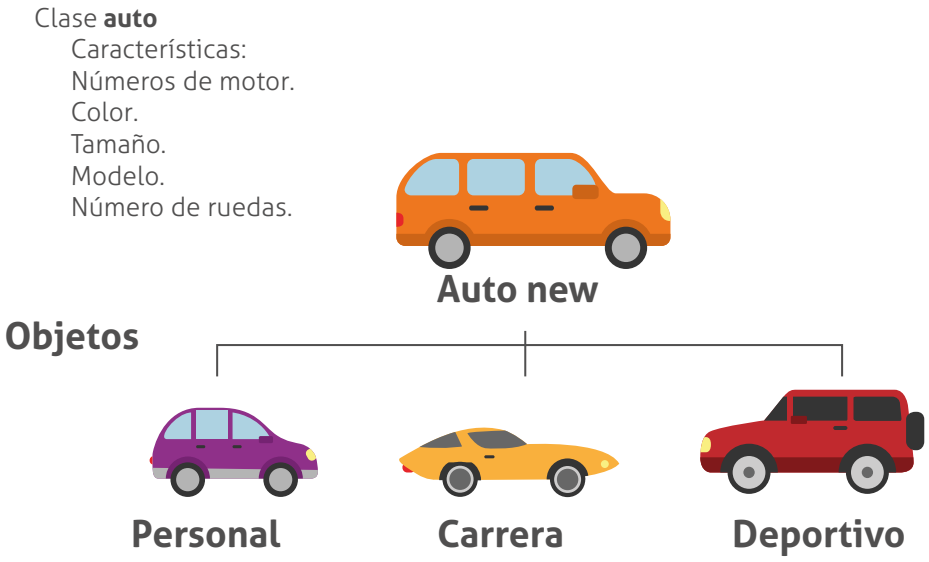


Figura 2. Clase Auto, instanciando tres objetos.
Fuente: Propia.

Los objetos se comunican entre sí, mediante el intercambio de mensajes, y contienen datos y operaciones.

Comportamientos o métodos de una clase: Son funciones que se definen dentro de la clase. Un objeto puede tener comportamientos propios y no ser compartidos con otros. En el ejemplo anterior, se definen como métodos de la clase auto: acelerar, frenar, detener, arrancar, girar, etc.; cada objeto que se crea de la clase auto,(personal, carrera, deportivo) van a compartir los mismos métodos

Metas y objetivos de la programación orientada a objetos

El objetivo de la POO es organizar, agrupar, catalogar mediante la construcción de clases que permitan visualizar de forma clara el código y el comportamiento de las aplicaciones, a través de la jerarquía de clases dependientes que se genera como un organigrama por niveles.

En forma general podemos tener una gran jerarquía de clases tal y como vemos en el siguiente gráfico. (Saco, 2014).

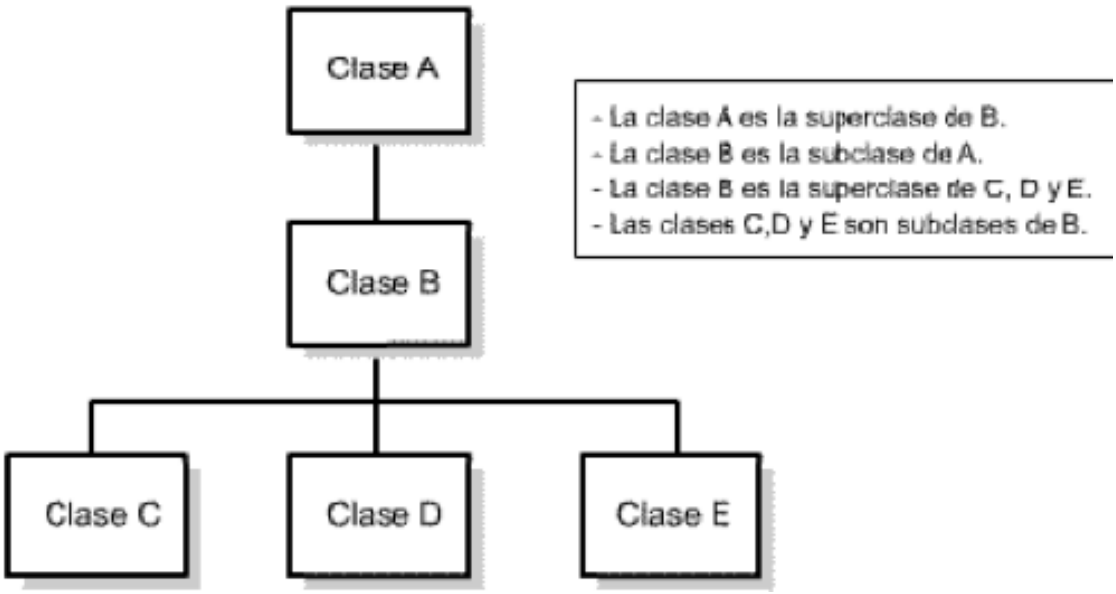


Figura 3.
Fuente: http://www.ciberaula.com/articulo/tecnologia_orientada_objetos/

Para encapsular las características y los comportamientos que los objetos presentan, lo haremos a través de la estructura "clase".

¿Qué es una clase?

Haciendo uso de la definición de clase vista anteriormente, el siguiente paso es aclarar para

qué sirve. Es una plantilla que permite definir atributos / variables y métodos / funciones.

Una clase contiene:

- Conjunto de atributos comunes
- Estructura de datos
- Comportamientos por medio de métodos

Para poder crear un objeto debemos instanciar una clase. Un grupo de objetos es representado por una clase, y tiene en común una estructura y unos comportamientos similares. Una clase define los atributos y los métodos necesarios y su encapsulamiento. Para acceder a sus datos se da de tres formas:

Pública: todos los datos y métodos de la clase podrán ser accesibles por otras clases.

Protegida: Solo la propia clase puede acceder a los datos y métodos definidos, y las demás clases solo pueden acceder cuando son derivadas o subclases de la clase protegida.

Privada: Solo puede ser accesible desde la misma clase.

Ejemplo de una clase en los tres estados publica, protegida y privada, definida en el lenguaje java.

```
publicclass Auto {
        números de motor
        color
        tamaño
        modelo
        numero de ruedas

Acelerar()
Frenar()
Girar()

}
```

```
privateclass Auto {
        números de motor
        color
        tamaño
        modelo
        numero de ruedas

Acelerar()
Frenar()
Girar()

}
```

```
protectedclass Auto {
        números de motor
        color
        tamaño
        modelo
        numero de ruedas

Acelerar()
Frenar()
Girar()

}
```

Figura 4. Clase en java en sus tres estados pública, privada y protegida
Fuente: Propia.

Propiedades de la Programación Orientada a Objetos

La POO presenta varias características, dentro del paradigma que definen diferentes lenguajes de programación. Estas son:

Abstracción: Es la propiedad de la POO, para tener en cuenta los aspectos más importantes de una situación y descarta los detalles mínimos, para así extraer los datos y métodos generales entre los objetos.

El nivel de abstracción está relacionado con la capacidad de interpretación que tiene el desarrollador para identificaren un problema las variables y objetos que intervienen,para posteriormente construir el algoritmo que da la solución al problema.

Encapsulado: Es la agrupación de datos y métodos que una misma clase, los objetos que contengan la misma características y comportamientos se agrupan en una clase.

Herencia: Es una propiedad que permite crear objetos derivando todas o algunas características y métodos o comportamientos definidos, en otras clases. Logrando así extender su funcionalidad.

Polimorfismo: Es lapropiedad que tienen los objetos de responder a un mensaje o un evento invocado; es la sobrecarga de parámetros, debido a que los objetos en sus características permiten sobrecargar diferentes parámetros para un solo método.

Los objetos en programación consisten en dividir en clases los objetos. Partiendo del concepto objeto se puede observar en el mundo actual variedad de objetos que tienen rasgos comunes o pueden ser completamente diferentes. Se pueden agrupar o relacionar de acuerdo a sus características y comportamientos del mundo real, como: tamaño, color, forma, etc.

Características de los objetos

Atributos:son las características propias de un objeto, estos se definen a través de variables que almacenan la información.

Métodos: son procedimientos o funciones que determinan las acciones que pueden realizar los objetos, ya sea sobre sí mismos o sobre otros objetos.

Lenguajes de programación orientados a objetos

Son los lenguajes que implementan los conceptos y técnicas de programación orientada a objetos.En esta unidad trabajaremos con Java como lenguaje de referencia y ejemplos de sintaxis. En la siguiente lista se mencionan algunos lenguajes de programación que soportan este paradigma.

- ABAP
- ABL Lenguaje de programación de OpenEdge de Progress Software
- ActionScript
- ActionScript 3
- Ada
- C++
- C#
- Clarion
- Clipper (lenguaje de programación) (Versión 5.x con librería de objetos Class(y))
- D
- Object Pascal (Delphi)
- Gambas
- Harbour
- Eiffel
- Java
- JavaScript (la herencia se realiza por medio de la programación basada en prototipos)

- Lexico (en castellano)
- Objective-C
- Ocaml
- Oz
- R
- Perl (soporta herencia múltiple. La resolución se realiza en preorden, pero puede modificarse al algoritmo linearization C3 por medio del módulo Class::C3 en CPAN)
- PHP (a partir de su versión 5)
- PowerBuilder
- Python
- Ruby
- Smalltalk (Proyecto investigativo. Influenció a Java.)
- Magik (SmallWorld)
- Vala
- VB.NET
- Visual FoxPro (en su versión 6)
- Visual Basic 6.0
- Visual Objects
- XBase++
- Lenguaje DRP Tomado del sitio web: (González, 2014)

Los métodos de una clase definen los comportamientos o acciones que realizan los objetos. A través de los métodos se intercambian mensajes entre los objetos que es su forma de comunicación.Cuando los objetos se comunican entre sí, lo hacen invocando un método de la clase; en java los métodos se invocan de la siguiente forma:

Nombre de objeto.nombre del método que se invoca.

Auto.acelerar();

Atuo.frenar();

Auto.girar();

El método permite desarrollar a través del código las sentencias que manipulan los datos. Desarrollar la lógica de programación permite invocar e intercambiar mensajes de un método a otro.

Cuando se define un método en java, su sintaxis consiste en: la forma de acceso, el nombre del método abre y cierra paréntesis y entre llaves "{}" se define el cuerpo o código del método. Ejemplo:

public acelerar (){

cuerpo o código del método;

Cuando definimos la clase Auto, con los métodos acelerar, frenar y girar, en los métodos se desarrolla el código que le permite a un Auto el proceso de acelerar.

Los métodos para la clase Auto se definen de la siguiente forma:

publicclass Auto

numeros de motor

color

tamaño

modelo

numero de ruedas

acelerar(){

```
        código_del_metodo_acelarar;
        }
        frenar(){
        código_del_metodo_acelarar;
        }

        girar(){
        código_del_metodo_acelarar;
        {
}
```

Veamos el primer ejemplo de programación orientada a objetos en java

Para realizar los programas en java se requiere de un IDE de desarrollo que permita integrar varias herramientas y poder escribir con facilidad el código de la aplicación.

Para realizar los ejercicios y talleres que se trabajarán durante este curso se hará uso del IDE de desarrollo Netbeans que puede ser descargado de su página oficial, y el Kit de desarrollo de java JDK que puede ser descargado de la página de Oracle.

Figura 5. NetBeans entorno de desarrollo.
Fuente: https://netbeans.org/

Figura 6. JDK, Kids de desarrollo java
Fuente: http://www.oracle.com/technetwork/es/java/javase/downloads/index.html

Para el proceso de instalación de la herramienta de NetBeans seguiremos los pasos que se especifican en la página oficial de NetBeans: https://netbeans.org/community/releases/69/install_es.html

Se realizará un ejercicio donde se explicará con detalle cada línea de la aplicación.

Realizar un programa que permita crear una clase que cargue el nombre y el género de una persona, mostrar los datos cargados, y dar como salida un mensaje si es hombre o mujer. Genero == Masculino / genero == Femenino

Programa:

```
importjava.util.Scanner;
public class Persona {
private Scanner teclado;
private String nombre;
private String genero;

public void inicio() {
teclado=new Scanner(System.in);
System.out.print("Ingrese nombre:");
nombre=teclado.next();
System.out.print("IngreseGenero:");
genero=teclado.next();
  }

publicvoid salida() {
System.out.println("Nombre:"+nombre);
System.out.println("Genero:"+genero);
  }

publicvoidesgenero() {
if (genero =="Masculino") {
System.out.print(nombre+" es Masculino.");
    } else {
System.out.print (nombre+" es femenino.");
}
  }
public static void main(String[] arg) {
Persona persona1;
    persona1=new Persona();
persona1.inicio();
persona1.salida();
persona1.esgenero();
  }
}
```

Análisis de la solución

En la POO cuando se crea una solución, el nombre de la clase a la que se hace referencia debe coincidir con el objeto del cual se hizo la abstracción, en este caso, la clase persona

```
publicclass Persona {
```

La primera línea

La palabra import permite importar dentro de la aplicación las librerías necesarias para ser usadas en cualquier momento; el método Scanner se usa para capturar datos por el teclado y es necesario importarla para las entradas del programa.

```
importjava.util.Scanner;
```

Los atributos se definen dentro de la clase y permiten el almacenamiento de los datos.

```
private Scanner teclado;
```

```
private String nombre;

privateint edad;
```

Los atributos pueden ser definidos como datos privados, que solo pueden ser accedidos por la clase que los genera.

Los atributos pueden ser accedidos por cualquier método o función de la clase, excepto el método main.

Después de definir los atributos de la clase, se procede a definir los métodos que interactúan en la aplicación. Ejemplo.

```
public void inicio() {

teclado=new Scanner(System.in);

System.out.print("Ingrese nombre:");

nombre=teclado.next();

System.out.print("IngreseGenero:");

genero=teclado.next();

  }
```

El método inicio es el primero que se debe invocar, pues se encarga de capturar las entradas por teclado, se define un objeto de la clase Scanner llamado "teclado", el objeto teclado usa el método next() para capturar los valores de nombre y género.

El siguiente método salida permite mostrar por consola un mensaje unido al nombre que tiene almacenado el atributo nombre de la persona.

La palabra void no retorna ningún valor para el método.

```
publicvoid salida() {

System.out.println("Nombre :"+nombre);

System.out.println("Genero :"+genero);

  }
```

El método género tiene una condición para evaluar si el género de la persona es masculino o de lo contrario es femenino. Y muestra un mensaje en consola de acuerdo a la condición de género == "Masculino".

```
publicvoidesgenero () {

if (genero =="Masculino") {

System.out.print(nombre+" es Masculino.");

    } else {

System.out.print (nombre+" es femenino.");

    }

  }
```

Finalmente se hace uso del método main que es el principal hilo de ejecución dentro del programa. Dentro de este método se crea una instancia del objeto persona, llamado persona1, que es el encargado de usar los métodos de dicha clase, en el orden que se defina.

```
public static void main(String[] arg) {

Persona persona1;

    persona1=new Persona();

persona1.inicio();

persona1.salida();

persona1.esgenero();
```

```
}
```

Persona persona1 se declara el objeto para su creación con la palabra reservada new.

persona1=new Persona(); se crea el objeto persona1 de la clase Persona

persona1.inicio(); se llama o invoca un método de la clase y lo hace con el objeto seguido del punto.

```
publicclass Auto {
        números de motor
        color
        tamaño
        modelo
        numero de ruedas

Acelerar()
Frenar()
Girar()

}
```

Forma correcta de definir la clase o plantilla

```
public class Auto {
        privateintnumerosdemotor;
        private String color;
        privateinttamano;
        privateint modelo;
        privatenumeroderuedas;

publicvoid acelerar() {
}

public void grenar() {
}
public void girar() {
}
public static void main(String[] arg) {
{
}
```

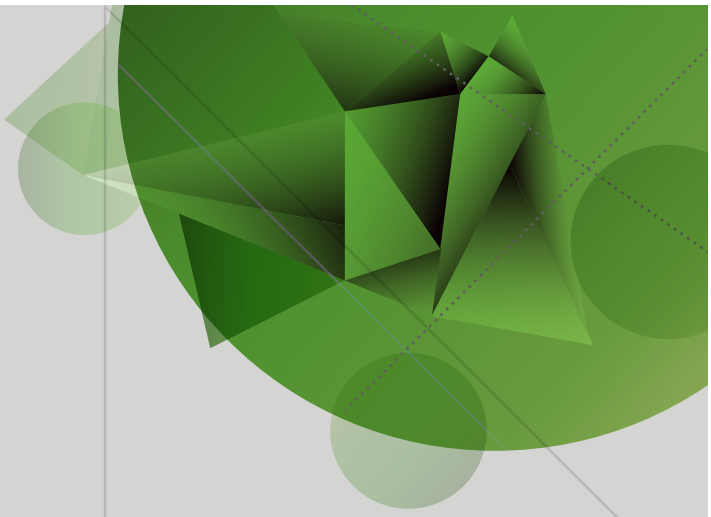
Retomamos el ejemplo inicial de esta semana de la clase Auto, con los atributos número de motor, color, tamaño, modelo, número de ruedas; los cuales son variables y deben pertenecer a un tipo de dato. l

Al observar la clase Auto, contiene cinco atributos comunes de cualquier auto, los cuales se deben definir dentro de la clase, sin espacios, sin tildes, ni ñ; cumpliendo así con los reglas para la definición de variables. Para conocer un poco más de cómo definir variables puede ampliar con el siguiente documento. 2-tiposdatoslenguajec.pdf que está en los recursos del programa.

2

Unidad 2

Características de la programación orientada a objetos



Modelos de programación I

Autor: Gustavo Enrique Tabarez Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El acceso a la información mediante el desarrollo de aplicaciones se puede realizar de tres formas, publica, privada y protegida, permitiendo controlar el acceso a los datos, el desarrollo de aplicaciones bajo java permite el control de los datos bajo estas tres formas de un clase, técnica que es utilizada por varios lenguajes de programación.

La información puede ser publica cuando se desea que todas los que deseen puedan consultarla en su momento, privada o protegida permite controlar el acceso.

Los objetos pueden tener información en estos tres tipos, y también recibir la información que se le comparta, haciendo esto que las aplicaciones sean más administrables.

U2

Metodología

Para cada unidad el estudiante cuenta con un documento, el cual le describe las actividades a realizar para la semana; dicho documento se denomina *guía de actividades.* Asimismo le informa las tareas calificables e incluye los objetivos de aprendizaje. Antes de iniciar usted debe leerlo muy atentamente.

La cartilla le permitirá abordar los temas a tratar en cada semana. Se recomienda leer cuidadosamente y en forma secuencial la información para tener una compresión adecuada de ellos.

De otro lado, cada semana cuenta con lecturas complementarias, recursos para el aprendizaje, actividades de repaso y videoconferencias que le permitirán aclarar dudas y/o profundizar en la temática semanal.

La actividad evaluativa le permitirá al estudiante evaluar su nivel de conocimiento referente al tema, y poder profundizar en aquellos que se encuentre más débil.

Encapsulamiento

El proceso de encapsulamiento en la POO, permite de forma oculta el poder cambiar la visualización de los elementos que componen la clase.El encapsulamiento se aplica para los atributos y los métodos de la clase, aplicando tres tipos de niveles de acceso:privados, protegidos y públicos.

Los atributos del objeto son envueltos por los métodos y la funciones que los operan en cada instrucción; los datos quedan situados en el centro de la clase cubiertos, pero respondiendo a las peticiones que se soliciten.

A este proceso de ocultamiento y protección de los datos de un objeto por medio de sus métodos, se define como encapsulamiento.

El encapsulamiento esconde, oculta detalles para otros objetos, que al ser llamados pueden variar en cualquier momento sin alterar las otras partes de la aplicación.

En Java, cuando se crea una clase, se puede controlar la forma de acceso y la visibilidad de los datos miembros y métodos al exterior de la clase, como se muestra en el siguiente ejemplo.

```
class A
{
 private int privx;
 protected int protb;
 public int pubc;
 int paqd;

 private void MetPriv() { ... }
 protected void MetProt() { ... }
 public void MetPub(A a)
{
 // Un campo siempre es visible
 // en la clase en donde se define
... privx ... a.privx ...
  ... protb ... a.protb ...
... pubc  ... a.pubc ...
 // Un método siempre es visible
 // en la clase en donde se define
  MetPriv(); a.MetPriv();
 }
 void MetPaq() { ... }
}
```

Ejemplo tomado de: (Mateu, 1996)

Privados: son los elementos que pueden ser accedidos solamente por la clase que los define, al ser privados pertenecen y son usados por la clase. La palabra reservada con la que se definen es private, en el lenguaje Java.

Protegidos: sus elementos pueden ser accedidos desde la clase que los define y de sus subclases.La palabra reservada para definirlas es protected en el lenguaje Java.

Públicas: sn elementos totalmente libres, y pueden ser accedidos por la clases que los define y las demás clases y métodos de la aplicación.La palabra reservada con la que se definen es public, en el lenguaje Java.

En el ejemplo anterior se puede ver que el dato miembro private int privx, es privado y solo puede ser accedido por la clase A.

El dato miembro protected int protb, es protegido por solo poder ser accedido por la clase A y sus métodos, y por las subclases que se definan de la clase A.

Los datos miembros public int pubc son públicos y pueden ser usados por toda la aplicación, el dato miembro int paqd, aunque no se definió como público, tomó el estado público.

El encapsulamiento en Java se da interno y externo.Cuando se definen elementos privados y protegidos son internos, y cuando se definen elementos públicos de la clase son externos:

El siguiente caso de forma más cotidiana nos muestra los estados de privado, protegido y público:

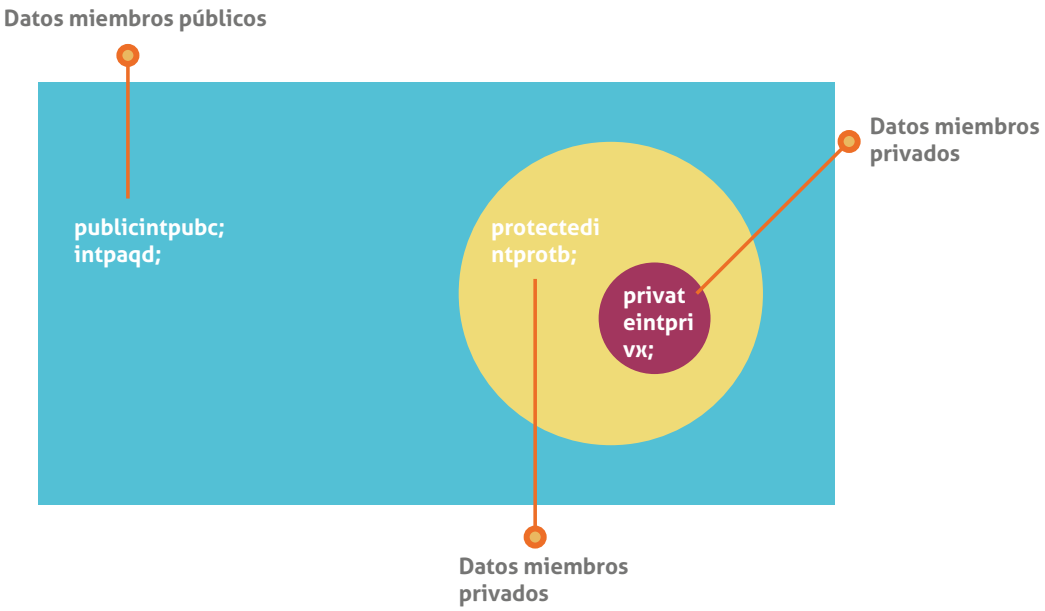


Figura 1. Estados de los datos miembros de una clase.
Fuente: Propia.

En una familia, un individuo tiene una información que solo la puede saber esa persona y es utilizada por él y nadie más. Este es caso de privado.

La misma persona adicional tiene una información que solo la puede compartir con los miembros de la familia y nadie más la puede saber por fuera del núcleo familiar, este es el caso de protegido.

Y hay una información que la puede saber cualquier amigo o persona por fuera de la familia sin ninguna restricción, este es el caso de público.

Veamos en ejemplo práctico en lenguaje java:

Se tiene el siguiente código, donde se crea una clase con un dato público, las cuales son accesibles desde fuera de la clase.

```
public class Uno {

public int valor;

}

class Acceso {

        public static void main (String[ ] args){


                Miclase obj = new Miclase();

                Obj.valor = -10;

        }

}
```

Al realizar un análisis del código se puede ver que se ha definido una variable de tipo entero "valor" y una clase pública, las cuales se pueden acceder directamente por medio de una instancia de la clase Uno, esto no presenta ningún error al momento de compilar, ¿qué sucedería si ingresamos un dato en la variable valor que sea diferente?, lo podríamos hacer sin ningún problema. Cuando protegemos el código, en este caso las variables, debemos crear un método que nos controle los valores que una variable pueda tener, y no tener acceso a ellas de forma directa. Aquí es donde aplicamos el encapsulamiento.

Para que un programa en java pueda tener flexibilidad y se pueda realizar un mantenimiento eficiente, el código desarrollado debe tener encapsulamiento de la siguiente forma:

- Se debe mantener las variables que se definan en la clase protegidas con private.
- Crear métodos públicos que accedan a las variables de forma directa.
- Usar los métodos de set y get para definir los métodos de ingresar y extraer.

El código anterior aplicando encapsulamiento:

```
public class Uno {

private int valor;

        public void setValor(int p){

        valor = p;

        }

        public int getValor(){

        return valor;

        }
```

```
}

class Acceso {

        public static void main (String[ ] args){


                Miclase obj = new Miclase();

                Obj.setValor(10);

                Systema.out.print("El dato de valor es"+obj.getValor());

        }

}
```

Principios de Modularidad

"Propiedad que tiene un sistema que ha sido descompuesto en un conjunto de módulos cohesivos y débilmente acoplados" (Booch, 1996).

El principio de modularidad entre objetos consiste en dividir la aplicación en módulos más pequeños, donde cada uno debe presentar un comportamiento tan independiente como sea necesario en toda la aplicación. La compilación de cada módulo se puede hacer por separado, pero dependiendo a través de conexiones con otros módulos, similar que en la encapsulación. En la programación orientada a objetos los lenguajes soportan la modularidad en muchas formas.

Divide y vencerás es un principio muy sabio, pues la modularidad en un sistema es la propiedad que permite crear una descomposición en subconjuntos relacionados por un acoplamiento entre ellos. Los programas modernos se crean por componentes diferentes y estos deben interactuar entre ellos de forma correcta.

En la figura No. 2, se muestra un programa dividido en cuatro objetos, que se relacionan entre sí, los objetos se comunican mediante el intercambio de mensajes y los módulos se comunican mediante interfaces bien definidas; también podemos ver la modularidad entre clases creando clases principales y subclases que se relacionen.

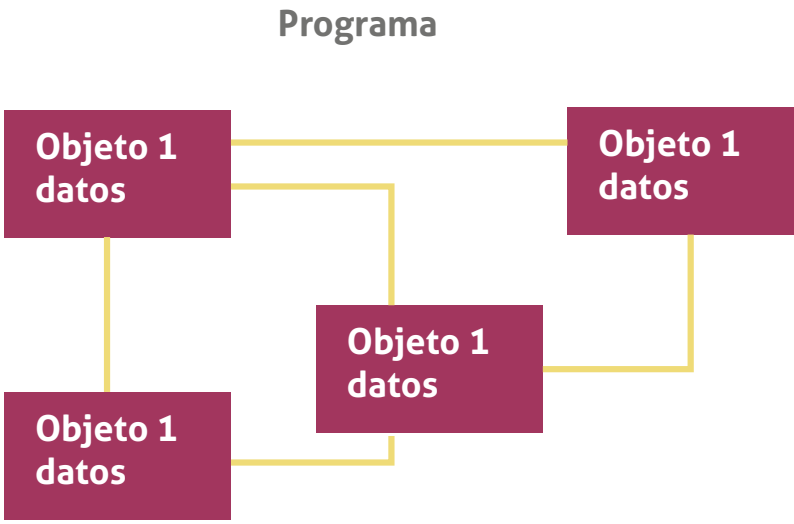


Figura 2. Modularidad y relación entre clases. Fuente: Propia.

Las relaciones que se pueden dar entre objetos son:

Tipo de relación	Descripción
Tiene un	El objeto de la clase tiene un atributo que son objetos de otras clases.
Usa un	Métodos pertenecientes al objeto de nuestra clase requieren del comportamiento de otras clases.
Es un	El objeto de nuestra clase es una extensión o especialización de la clase de la cual hereda.

Tabla 1. Tipo de relaciones
Fuente: (Nacional, 2006)?

Características de la Modularidad

Alta cohesión: es un módulo que tiene responsabilidades altamente definidas y se relaciona con otros, sin hacer mayor protagonismo.

Bajo acoplamiento: es un módulo que tiene poca dependencia de otros módulos. Facilita la **comprensión modular:** se puede entender un módulo con facilidad sin conocer los otros.

Continuidad modular: Al realizar un cambio en la aplicación, este solo afecta a un módulo especifico o a muy pocos.

Protección modular: El resultado en una situación inesperada generada por un módulo, afecta solo al módulo o a unos pocos.

La estructura de un módulo se da mediante una interfaz, y su implementación está conformada por una estructura de conjunto de operaciones y atributos bien definidos.

Interfaz e implementación: constituyen un módulo altamente integrado donde la interfaz son operaciones que definen el comportamiento de la aplicación y la implementación son las sentencias de código, las clases, objetos y módulos que estructuran la aplicación. El ejemplo del iceberg muestra solo una parte en la superficie del agua, mientras su gran estructura se encuentra bajo la superficie y no es visible a simple vista.

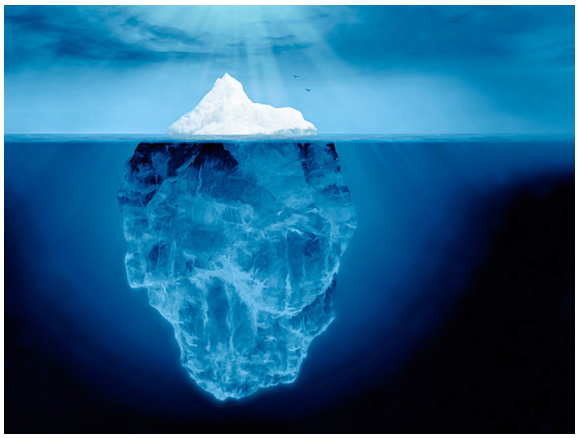


Figura 3. Tamaño de un iceberg.
Fuente: www.taringa.net

Ejemplo de modulación y encapsulamiento en Java:

```
public class Circulo1 {
  private float radio;
  private float perimetro;
  private float area;
  public float getRadio() {
    return radio;
  }

  public void setRadio(float radio) {
    this.radio = radio;
  }

  public float getPerimetro() {
    return perimetro;
  }
  public void setPerimetro(float perimetro) {
    this.perimetro = perimetro;
  }
  public float getArea() {
    return area;
  }
  public void setArea(float area) {
    this.area = area;
  }

  public void calculeArea() {
     area = (float) (Math.PI * Math.pow(radio, 2));
  }

  public void calculePerimetro() {
    perimetro = (float) (Math.PI * (2 * radio));
  }
  public void imprimeArea() {
    JOptionPane.showMessageDialog(null, "El area es" + area);
  }

  public void imprimePerimetro() {
    JOptionPane.showMessageDialog(null, "El perimetro es" + perimetro);
  }
  public void lee() {
       radio = Float.parseFloat(JOptionPane.showInputDialog(null, "Radio :"));
}
}
```

En el código anterior se ha definido una clase pública llamada Circulo1, con tres datos privados: radio, perímetro, área, y unos métodos getRadio(), SetRadio(), getPerimetro(), setPerimetro(), getArea(), que nos permitirán manipular mediante variables locales los datos privados de la clase, y no acceder directamente a ellos.

La clase AplicaciónCirculo se ha define como una clase public con un solo método main, donde se instancia la clase Circulo1 para crear el objeto c1, y así poder acceder a los métodos que tiene la clase Circulo1.

```
public class AplicacionCirculo {

  public static void main(String[] args) {

Circulo1 c1;

    c1 = new Circulo1();


    c1.lee();

    c1.calculeArea();

    c1.imprimeArea();

    c1.calculePerimetro();

    c1.imprimePerimetro();

  }

}
```

El método Get y Set

Los métodos get y set, son métodos utilizados en una clase para desplegar y capturar atributos.Las variables pertenecientes a una clase por lo general son privadas y solo se pueden acceder por los objetos que la contengan. Los métodos getters y setters, son métodos definidos en java para interactuar con las variables privadas de una clase.

Un método get retorna el valor de una propiedad de un objeto sobre la variable miembro de un objeto.

Un método set tiene la función void como un tipo de retorno, tomando un parámetro sobre una variable para luego asignársela a la variable miembro de la clase.

Dentro de las convenciones en java, los métodos set y get son utilizados por los métodos externos, y por lo cual, estos se crean como métodos públicos, totalmente visibles.La convención definida es definir métodos get y set para las variables miembros asociadas. Ejemplo getCodigo, y setCodigo, asociado a la variable Código. No todas las variables miembros van a tener definidos los métodos get y set como acceso asociado. Solo se definen estos métodos para las variables que van a ser accesadas externamente. En la figura se muestra la creación de los métodos get y set a través del IDE de desarrollo eclipse.

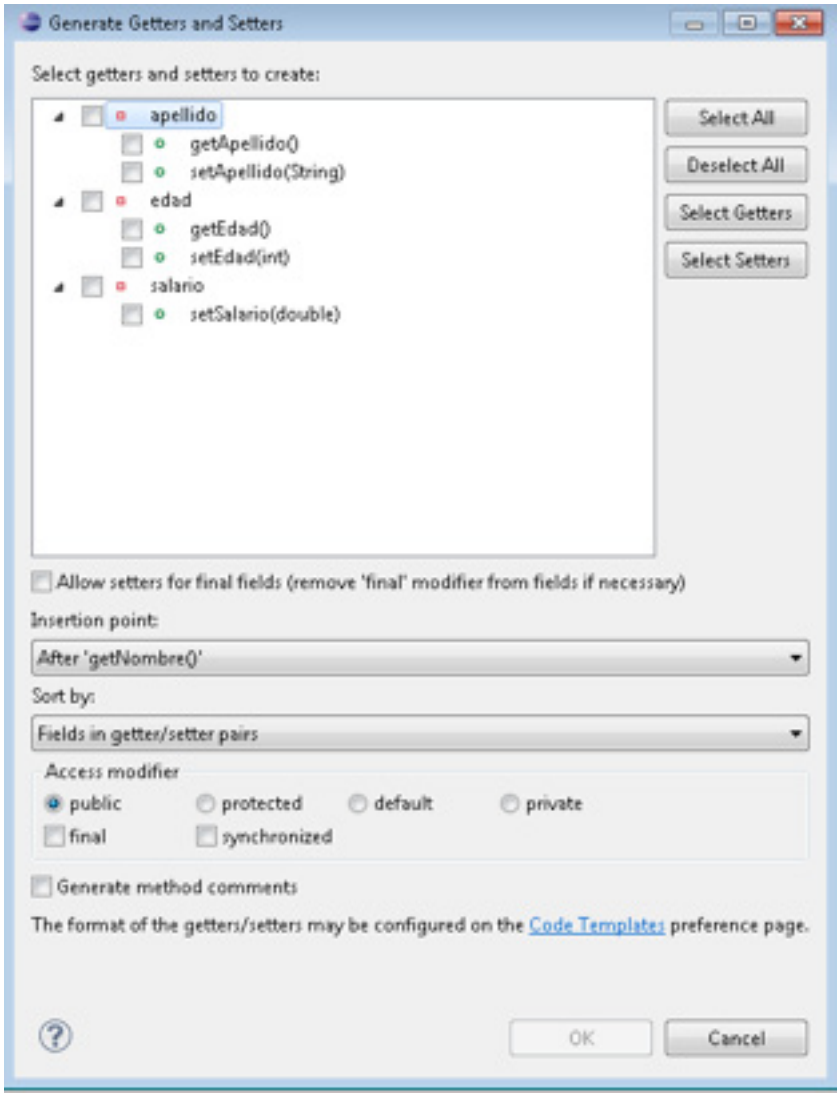


Figura 4. Creación de métodos get y set, para tres variables miembros de la clase.
Fuente: Propia.

Ejemplo, donde se aplicará el encapsulamiento y modularidad definiendo los métodos get y set para acceder a variables miembros de una clase. Se definirá una clase llamada persona para crear los métodos get y set.

public class Persona( ){

private String nombre;

private String apellidos;

private int edad;

private boolean genero; //Tomaremos que el valor verdadero para Masculino, y el falso Femenino.

```
 public Persona(){ } // creamos el constructor

 // Creamos los métodos gets y sets de la clase.

public String getNombre(){
   return this.nombre;
 }

 public void setNombre(String nombre){
this.nombre = nombre;
 }

public String getApellidos(){
   return this.apellidos;
}

 public void setApellidos(String apellidos){
   this.apellidos = apellidos;
}

 public int getEdad(){
   return this.edad;
 }

 public void setNombre(String edad){
  this.edad= edad;
 }

 public boolean getGenero(){
  return this.sexo;
 }

 public void setGenero(String boolean ){
this.boolean = boolean;
 }
}
```

Como se puede visualizar en el código, los métodos lo que hacen es proporcionar el acceso a los datos miembros de la clase que son privados.

Para usar la clase dentro de otra clase y hacer uso de los métodos get y set, se hacede la siguiente forma:

```
Persona persona = new Persona( );

persona.setNombre("Carlos");

persona.setApellidos("Torres");

persona.setEdad(27);

persona.setSexo(true);

String nombreCompleto = persona.get-Nombre( ) + " " + persona.getApellidos( );
```

Con este código, se estarían pasando datos a los métodos para ser asignados a los datos miembros privados de la clase.

Ejercicio: realizar una aplicación donde se evidencie el encapsulamiento, modularidad y los métodos de acceso a los miembros privados a una clase, para capturar datos de un equipo de cómputo, con los datos miembros de la clase: código, nombre, precio y cantidad.

Conceptos de interfaz y objeto

Las interfaces en java son un conjunto de métodos sin implementación, solo definen de forma similar a los métodos abstractos.

Las interfaces son tipos de referencia, no se pueden crear objetos de ellas, solo se crean tipos que se derivan de ellas y se aplica el polimorfismo.

Se pueden definir tipos que desciendan de más de una interfaz, debido a los problemas que se presentan cuando se hereda de varias clases.Con la definición de interfaces no se presenta este conflicto, tanto que no incluyen código, se logra la herencia múltiple de las mismas.

Las estructuras no heredan de las clases pero sí de interfaces, y las interfaces no pueden derivar de clases, pero sí de otras interfaces. (Seco, 2006).

Por ejemplo, un alumno que sabe francés, conocemos lo que es capaz de hacer, además de ser persona se aplica la herencia, el alumno cumple con la interfaz de intérprete de francés. Además podríamos decir que también él es un intérprete del idioma francés.

Interfaces

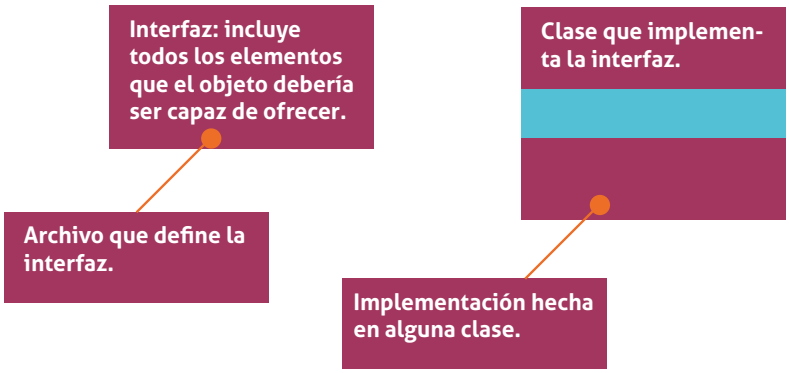


Figura 5. Implementación de una interfaz. Fuente: Propia.

Las interfaces solucionan el problema de herencia múltiple que no se presenta en Java, lo que se pretende es definir un elemento donde se pueda compartir las características comunes entre las clases de una aplicación, con esta implementación se busca controlar los métodos y atributos para que estén disponibles para las clases que las implementen.

Ejemplo de interface en java

A continuación se presentará un ejemplo simple de definición y uso de interface en Java. Las clases que se van a usar y sus relaciones, se muestran en el esquema. Escribe el código y ejecútalo.

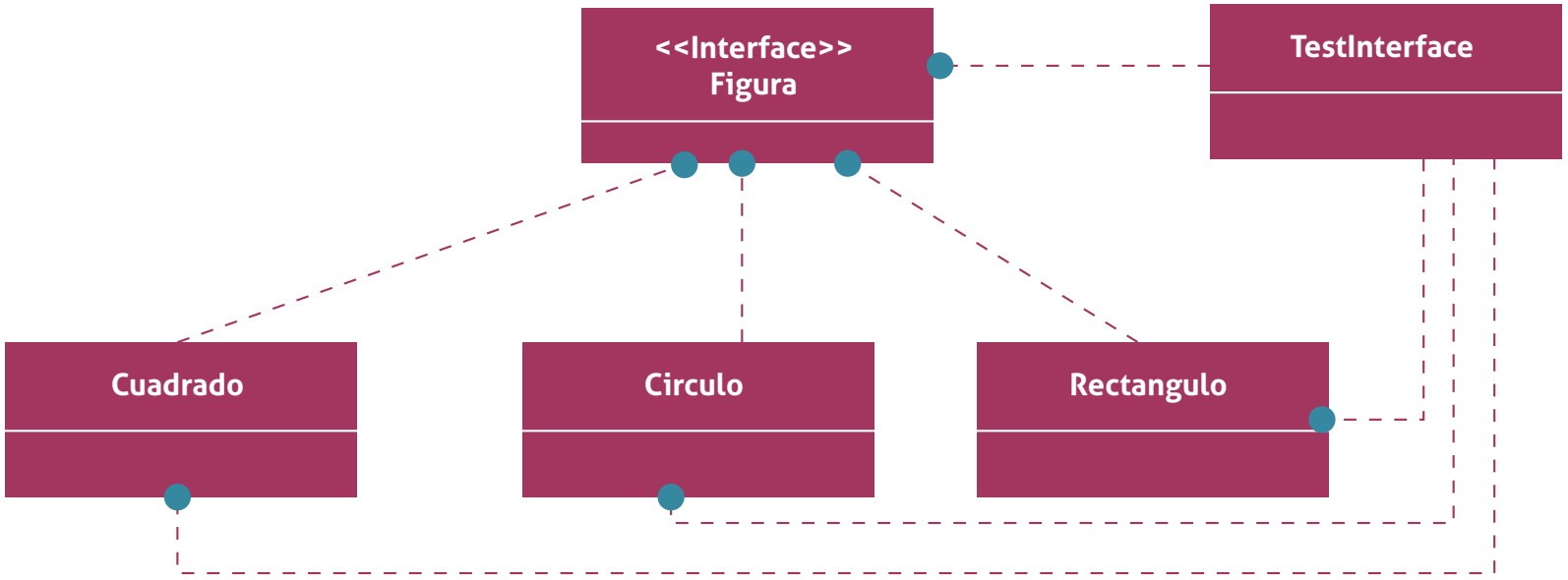


Figura 6. Diagrama de interfaces.
Fuente: http://aprenderaprogramar.com/index.php?option=com_content&view=article&id=671:para-que-sirven-las-interfaces-java-implementar-una-interfaz-del-api-ventajas-y-ejemplos-basicos-cu00697b&catid=68:curso-aprender-programacion-java-desde-cero&Itemid=188

```
public interface Figura {
float PI = 3.1416f;  // Por defecto public static  final. La f final indica que el número es float
float area();     // Por defecto abstract public
} //Cierre de la interface

public class Cuadrado implements Figura { // La clase implementa la interface Figura
private float lado;
public Cuadrado (float lado) { this.lado = lado; }
public float area() { return lado*lado; }
}
public class Circulo implements Figura{ // La clase implementa la interface Figura
private float diametro;
public Circulo (float diametro) { this.diametro = diametro; }
public float area() { return (PI*diametro*diametro/4f); }
}
```

```
public class Rectangulo implements Figura{ // La clase implementa la interface Figura

private float lado; private float altura;

public Rectangulo (float lado, float altura) { this.lado = lado; this.altura = altura; }

public float area() { return lado*altura; }

}

import java.util.List; import java.util.ArrayList;

public class TestInterface {

   public static void main (String [ ] Args) {

Figura cuad1 = new Cuadrado (3.5f); Figura cuad2 = new Cuadrado (2.2f); Figura cuad3 = new
Cuadrado (8.9f);

         Figura circ1 = new Circulo (3.5f); Figura circ2 = new Circulo (4f);

         Figura rect1 = new Rectangulo (2.25f, 2.55f); Figura rect2 = new Rectangulo (12f, 3f);

        List <Figura> serieDeFiguras = new ArrayList <Figura> ();

        serieDeFiguras.add (cuad1); serieDeFiguras.add (cuad2); serieDeFiguras.add (cuad3);

serieDeFiguras.add (circ1); serieDeFiguras.add (circ2); serieDeFiguras.add (rect1); serieDeFigu-
ras.add (rect2);

         float areaTotal = 0;

         for (Figura tmp: serieDeFiguras) {     areaTotal = areaTotal + tmp.area();    }

 System.out.println ("Tenemos un total de " + serieDeFiguras.size() + " figuras y su área total es
de " +

        areaTotal + " uds cuadradas")  } } //Cierre del main y de la clase
```

El resultado de ejecución podría ser algo así:

Se tiene un total de 7 figuras y su área total es de 160.22504 uds cuadradas. En este ejemplo se **comprueba que la interface Figura define un tipo.** Podemos crear un ArrayList de figuras donde tenemos figuras de distintos tipos (cuadrados, círculos, rectángulos) aprovechándonos del polimorfismo. Esto nos permi-

te darle un tratamiento común a todas las figuras. En concreto, usamos un bucle foreach para recorrer la lista de figuras y obtener un área total.

Metaclases: La parametrización que se le da a una clase cuando se relaciona con otra es muy flexible para poder crear clases que puedan se reutilizables.

Relación de metaclases:a una clase se le llama metaclase, cuando sus atributos son objeto de otra clase.

La instanciación y parametrización: es poder llegar a tener clases que sean reutilizables.Relación que se da entre clases y objetos.En la aplicación las clases son completamente estáticas, hacen parte del programa y no presentan ningún cambio durante su ejecución.Por el contrario, los objetos son dinámicos, pues estos se crean, cambian de estado durante su ejecución y al final se destruyen.

Clasificación

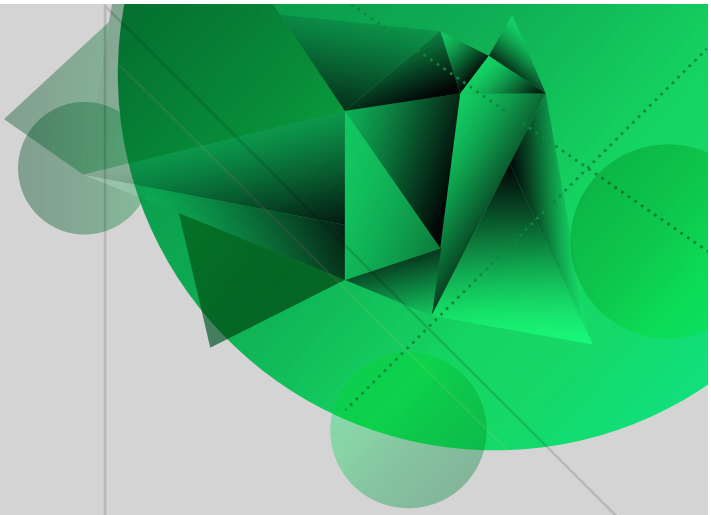
Forma de ordenar el conocimiento. No existen reglas simples que permitan la mejor selección de clases y objetos para generar una clasificación. Clasificar es agrupar cosas que tienen estructura común o comportamiento similar. La clasificación es un proceso difícil, por lo que conviene hacerlo de manera incremental iterativa.

Al iniciar el diseño, se define una estructura inicial para las clases. En base a la experiencia adquirida se crean nuevas subclases a partir de las existentes o bien es muy común que se requiera dividir una clase en otras más pequeñas (pero más generales), inclusive es posible requerir de la unión de varias clases. (Maldonado, 2008).

3

Unidad 3

Herencia



Modelos de programación I

Autor: Gustavo Enrique Tabarez Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

En esta unidad el estudiante comprenderá la funcionalidad de la herencia en la programación orientada a objetos, la jerarquía de clases que se da cuando se tiene clases padres y clases hijas.

Mediante ejemplos explicados se mostrará la implementación de herencia simple en java y la dependencia de clases.

Se verá el concepto de herencia múltiple y la deficiencia que tiene java en la implementación de herencia múltiple.

U3 Metodología

Para cada unidad el estudiante cuenta con un documento texto, el cual le describe las actividades a realizar para la semana; dicho documento se denomina Guía de actividades. Asimismo le informa las tareas calificables e incluye los objetivos de aprendizaje. Antes de iniciar usted debe leerlo muy atentamente.

De otro lado, cada semana cuenta con lecturas complementarias, recursos para el aprendizaje, actividades de repaso y videoconferencias que le permitirán aclarar dudas y/o profundizar en la temática semanal.

La actividad evaluativa, le permitirá al estudiante, evaluar su nivel de conocimiento referente al tema y poder profundizar en aquellos contenidos que se encuentre más débil.

Para esta semana hay un material de apoyo videocápsula, videodiapositiva, que le permitirá afianzar los conceptos y comprender la herencia que se da en la programación orientada a objetos.

Herencia

Dos o más clases pueden estar relacionadas por la colaboración entre clases, otro tipo de relación que se da en la programación orientada a objetos es la herencia entre objetos.

La herencia ordena las clases formando estructuras jerárquicas, logrando una jerarquía entre las clases de varios niveles. Ejemplo:

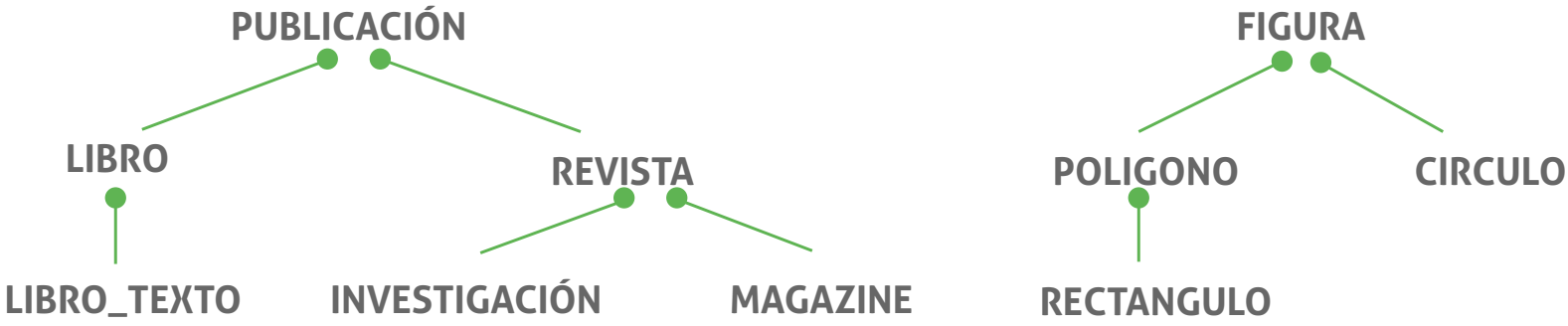


Figura 1.
Fuente:Propia.

Si la clase "LIBRO" hereda de la clase "PUBLICACION" entonces: la clase "LIBRO" incorpora la estructura de atributos y métodos de la clase "PUBLICACION".

La clase "LIBRO" puede incluir cambios y adaptaciones a su clase. "LIBRO" puede añadir nuevos atributos, nuevos métodos, y redefinir métodos.

La herencia es la característica que más se presenta en la POO, método por el cual se puede reutilizar el código.

La herencia permite hacer uso de atributos y métodos de la clase superior que la deriva, como si las dos fueran una sola clase.

Ejemplo:

Herencia simple: se creará una clase padre que heredará el apellido y otra clase hija donde

se asigna el nombre.

```
packageherencia;

public class ClaseHeredada{

String Apellido;

publicClaseHeredada (String Dato){

                this.Apellido = Datos;

}

}
```

Una vez creada la clase padre, procedemos a crear la clase hijo con el nombre de la persona:

```
package herencia;

publicclass Herencia extendsClaseHeredada{

        String Nombre;

        public Herencia(String Texto){

                super(Texto);

}

public void setPersona(String NombrePer){

this.Nombre = NombrePer+this.Apellido;

}

public String getPersona(){

returnNombre;

}

}
```

La palabra extends indica el programa que se está heredando de la clase "Padre" en donde está el apellido. La palabra super se usa para instanciar al constructor de la clase que está heredando, con los métodos setPersona y getPersona, son los métodos para capturar y posteriormente mostrar el resultado.

Luego se proceda instanciar la clase para hacer uso de los atributos y métodos de la clase "Padre", en el siguiente código se muestra como se instancia la clase:

public static void main(String [] args){

Herencia X = new Herencia("Pedro Perez");

X.setPersona("Carlos Torrez");

System.out.println(X.getPersona());

}

Instanciamos la clase "herencia" pasándole internamente al constructor el apellido, el cual lo pasará a la clase principal "ClaseHeredada", le asignamos el nombre haciendo uso de su método setPersona, lo mostramos por pantalla a través de la función System. out.println y el método getPersona. (Figueroa, 2012).

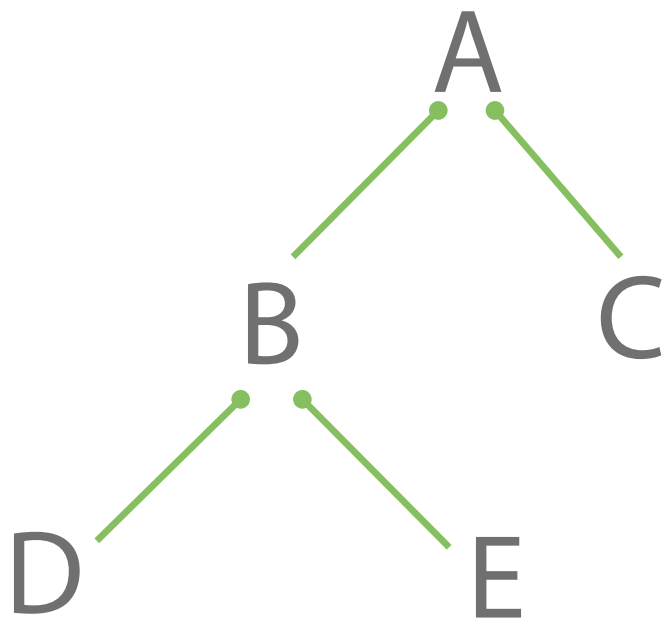


Figura 2.
Fuente: Propia.

Algunas ventajas de la herencia en la Programación Orientada a Objetos

El ahorro de código es bastante, se puede heredar atributos y métodos de otras clases.

Permite la reutilización del código extendiendo su funcionalidad de la clase principal.

Desventajas

Se crea una fuerte dependencia en la clase "Hija" con relación a la clase "Padre".

Tipos de herencia

Herencia Simple

Cuando una clase puede heredar de una única clase. Como muestra la figura.

Superclases y subclases

La estructura jerárquica que se da en la programación orientada a objetos con la aplicación de herencia entre clases, muestra la dependencia que tiene las clases de niveles inferiores sobre las clases "Padre".

En el ejemplo que muestra la figura cada clase tiene una dependencia de la clase "Padre". Estas son denominadas superclases y las clases que derivan son llamadas subclases.

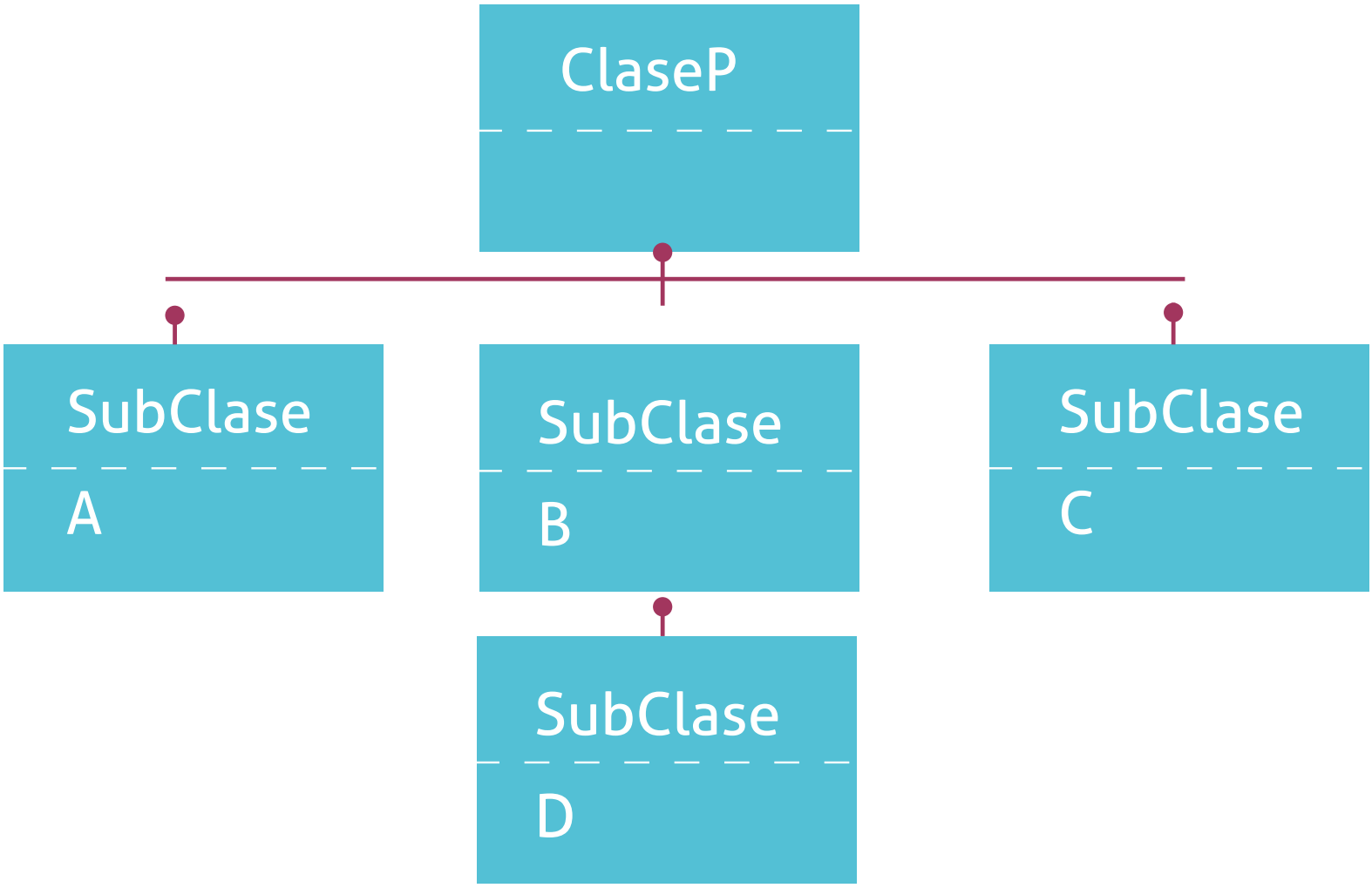


Figura 3.
Fuente: Propia.

La ClaseP es la clase padre, clase principal o superclase de la subclaseA, subclaseB, subclaseC.

La SubclaseB es la superclase de la subclaseD.

Las subclases heredan de forma automática los atributos y métodos de la superclases, adicionalmente una subclase puede adicionar atributos propios y métodos propios para su funcionalidad que la superclase no tiene.

En java para crear una subclase se usa la palabra **"extends"** en la creación de la clase. Ejemplo:

*Classnombresubclase**extends**nombresuperclase{*

}

Extends: significa que la subclase extiende de la superclase, cuando una clase no tiene una clase padre para heredar, por defecto en java la clase global es la clase object.

Jerarquía de herencia

- **Generalización:** se da cuando se detectan o existen clases con un comportamiento en común. Por ejemplo, un libro y una revista ambas son publicaciones, en esta jerarquía vemos la factorización.
- **Especialización:** se da cuando una clase es un caso especial de otra. Por ejemplo, un rectángulo es un tipo de polígono, en esta jerarquía vemos la abstracción:
- **Ejemplo de herencia en Java:** mediante el siguiente ejercicio, se aplicará el concepto de herencia.

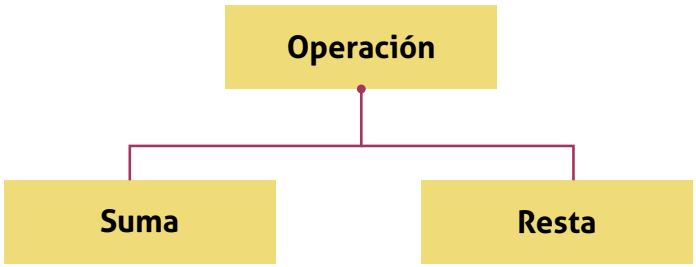


Figura 4.
Fuente: Propia.

El método operación es diferente para las clases "Suma" y "Resta", por lo tanto, no podemos disponer de la clase "Operación". Los métodos que permiten asignar un valor a los atributos asignar1, asignar2, y mostrar resultados, serán iguales en las dos clases, esto nos permite que podamos accederlos en la clase "Operaciones". Igualmente los atributos de dato1 y dato2 y resultado, lo definiremos en la clase padre "Operacion".

En el IDE de desarrollo crea un proyecto con cuatro clases: Operacion, Sumar, Restar y Ejecucion.

```
import java.util.*

public class Operacion {

protected Scanner teclado;

protectedint dato1;

protectedint dato2;

protectedintresultado;

publicOperacion(){

teclado = new Scanner(System.in);

}

publicvoid asignar1(){

        System.out.println("Ingresar el primer dato : ");

        dato1 = teclado.nextInt();

}

publicvoid asignar2(){

System.out.printl("Ingresar el segundo dato :");
```

```
valor2 = teclado.nextInt();
}
publicvoidmostrarresultados(){
        System.out.printl(resultado);
        }
}
public class Suma extends Operacion {
        publicvoid realizar(){
                resultado = dato1 + dato2;
        }
}
publicclass Resta extendsOperacion {
        publicvoid realizar (){
                resultado = dato1 - dato2;
        }
}
public class ejecucion (){
public static void main (String[] args){
Suma sumar = new Suma();
sumar.asignar1();
sumar.asignar2();
sumar.realizar();
System.out.println("El suma es  : ");
Sumar.mostrarresultado();
Resta restar = new Resta();
restar.asignar1();
restar.asignar2();
restar.realizar();
Systema.out.println("La resta es: ");
Restar.mostrarresultado();
}
}
```

En el ejercicio la clase "Operacion" tiene 4 atributos o variables.

```
Import java.util.*
public class Operacion {
private Scanner teclado;
privateint dato1;
privateint dato2;
privateintresultado;
```

Los atributos en la clase son definidos como protected, para que las clases que heredan de la clase principal, es decir, las subclases puedan utilizar estas variables sin problema.

Los métodos que se definieron en la clase "Operacion"son:

```
publicOperacion(){
teclado = new Scanner(System.in);
}
```

```
publicvoid asignar1(){

        System.out.println("Ingresar el primer dato : ");

        dato1 = teclado.nextInt();

}

publicvoid asignar2(){

System.out.printl("Ingresar el segundo dato :");

valor2 = teclado.nextInt();

}

publicvoidmostrarresultados(){

        System.out.printl(resultado);

        }
```

En el primer método se crea un constructor para activar el método de captura por teclado.

El segundo y tercer método capturan los valores en dato1 y dato2 para realizar la operación.

El cuarto método muestra los resultados de la operación.

Para heredar de la clase padre a la clase hijo, se realiza de acuerdo a la siguiente instrucción:

```
publicclass Suma extendsOperacion
```

Con la palabra extends y el nombre de la clase padre "Operacion", lo que indica que los métodos que están definidos en la clase padre, pueden ser accedidos sin ninguna restricción por la clase hijo "Suma".

La clase "Suma" tiene su propio método que es único para su clase, el cual es llamado:

```
publicvoid realizar (){

                resultado = dato1 + dato2;

        }
```

Encargado de realizar la suma y la resta de la clase "Operacion".

El método realizar accede a la clase padre y utiliza los atributos heredados, que fueron definidos como protected, de otra forma si hubiesen sido definidos como private, la clase hijo no podría acceder a ellos.

La clase "Suma" tendría 5 métodos, uno propio y cuatro métodos heredados. Luego en la clase "Ejecucion" creamos el objeto de la clase "Suma", para acceder a sus atributos y métodos heredados.

```
public class ejecucion (){

public static void main (String[] args){

Suma sumar = new Suma();

sumar.asignar1();

sumar.asignar2();

sumar.realizar();

System.out.println("El suma es  : ");

Sumar.mostrarresultado();

Resta restar = new Resta();

restar.asignar1();

restar.asignar2();
```

restar.realizar();

Systema.out.println("La resta es: ");

Restar.mostrarresultado();

}

}

Los métodos son accedidos por el objeto **sumar,** quien instancia al método de la clase **"Realizar"** y a los métodos heredados, el cual debe conocer que métodos son públicos para utilizarlos.

De igual forma pasa con la clase "Resta", que tiene el mismo comportamiento que la clase "Suma", con la diferencia en el signo.

La clase "Operacion" tiene los métodos y atributos comunes, por lo tanto, no se hace necesario definir un objeto para esta clase.

Herencia múltiple

Cuando una clase hereda de varias clases. Formando un grafo dirigido a cíclico.

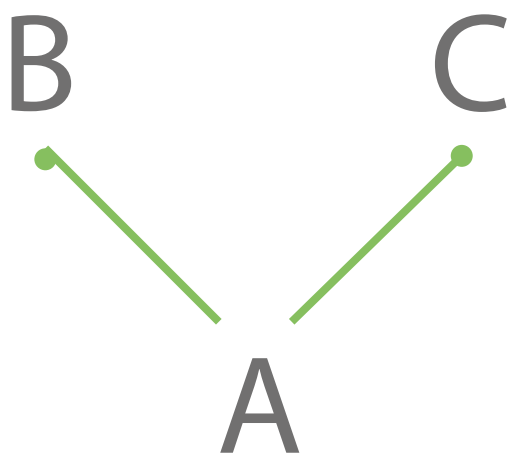


Figura 5.
Fuente: Propia.

En la figura se representa la subclase A, que hereda de la superclase B y la superclase C, heredando de dos clases sus atributos y sus métodos, aplicando herencia múltiple. La subclase A tiene además sus propios atributos y métodos únicos de su clase.

La herencia múltiple, tiene un alto grado de complejidad, de allí que muy pocos lenguajes lo permitan.

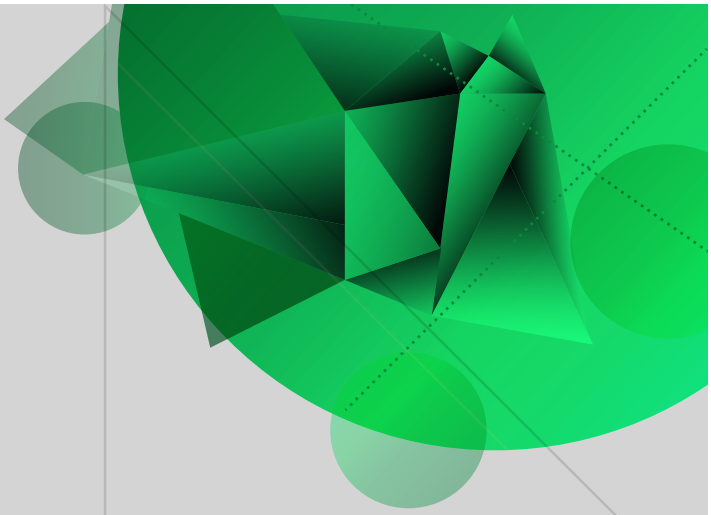
"Java no soporta la implementación de la herencia múltiple, para esta deficiencia tiene la implementación de interfaces, que logra con esto simular de forma limitada la implementación de la herencia múltiple en java". (Rodríguez, 2006).

Las interfaces es tema de la próxima unidad.

3

Unidad 3

Interfaces en java



Modelos de programación I

Autor: Gustavo Enrique Tabarez Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

La programación en java tiene unas grandes ventajas en el desarrollo de aplicaciones que están bajo la tecnología orientada a objetos e incluso presenta varias tecnologías de desarrollo, como el desarrollo web, móvil, el desarrollo de aplicaciones empresariales. Con todo esto, estas tecnologías, presentan inconvenientes a la hora de implementar la herencia múltiple.

Para esto, java implementó el manejo de interfaces de programación, nada que ver con la programación gráfica del manejo de formularios, son dos cosas totalmente diferentes.

La interface permite de alguna forma solucionar en gran medida la deficiencia de la herencia múltiple en el lenguaje de programación en java, la cual permite agrupar métodos y atributos en una interface. Las interfaces en java no son clases, simplemente son agrupaciones de métodos y atributos.

U3 Metodología

Para cada unidad el estudiante cuenta con un documento, el cual le describe las actividades a realizar para la semana; dicho documento se denomina Guía de actividades. Asimismo le informa las tareas calificables e incluye los objetivos de aprendizaje. Antes de iniciar usted debe leerlo muy atentamente.

Se recomienda seguir en un orden secuencial la lectura y estar muy atento al código que se utiliza para su desarrollo. En cada ejercicio que se expone en la semana, encontrará la explicación detallada para una mayor comprensión del tema. Se recomienda realizar ejercicios con base en los ejemplos que están desarrollados para comprender mucho más fácil el concepto de interface en java.

De otro lado, cada semana cuenta con lecturas complementarias, recursos para el aprendizaje, actividades de repaso y videoconferencias; que le permitirán aclarar dudas y/o profundizar en la temática semanal.

La actividad evaluativa, le permitirá al estudiante medir su nivel de conocimiento referente al tema y poder profundizar en aquellos tópicos donde se vea más deficiente.

Interfaces

Una interfaz en java es una agrupación de métodos abstractos y de propiedades. En las interfaces no se define la implementación de los métodos; para tal efecto, se hace a través de clases que describen la lógica de los métodos.

Cuando se trabaja con interfaces en java se soluciona de cierta forma la debilidad que presenta java al implementar la herencia múltiple. Al definir interfaces se definen variables polimórficas y la ejecución de métodos polimórficos.

El papel fundamental al definir interfaces en java es la separación en la clase con relación a qué hace, y cómo lo hace. Lo que ha permitido construir programas mucho más eficientes y más complejos, a continuación se muestra un ejemplo: (Rodríguez, 2009).

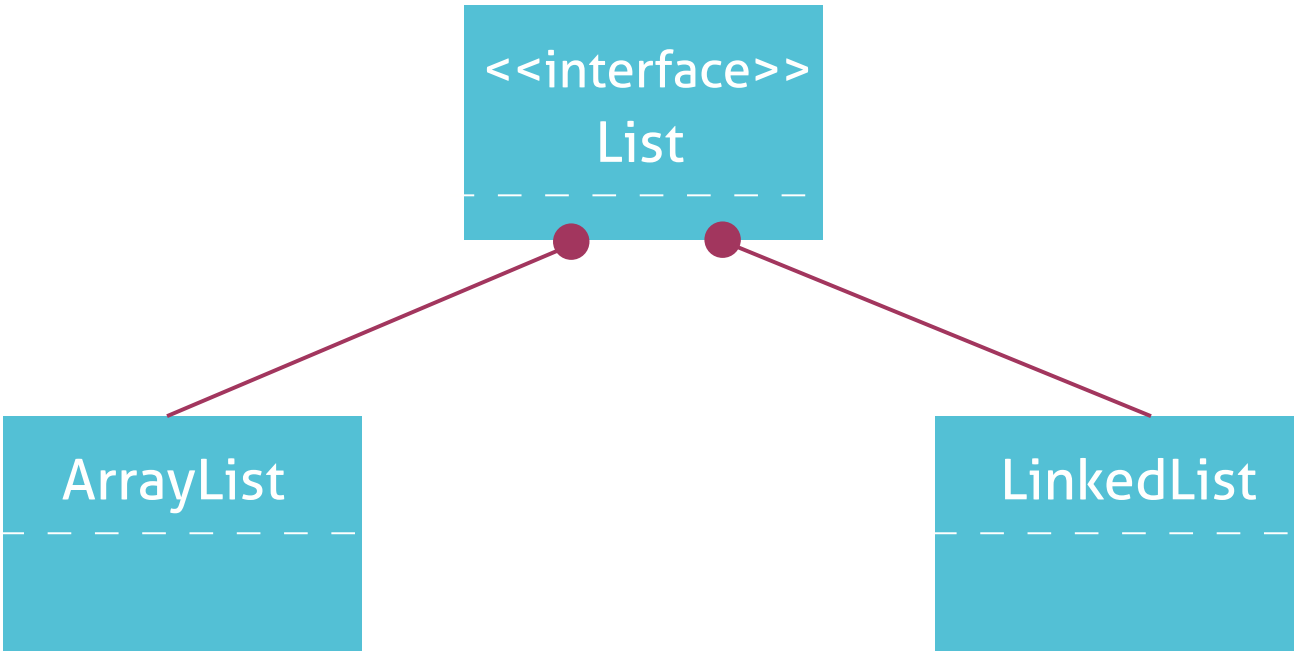


Figura 1. Interfaz List, que es implementada por las clases ArrayList y LinkedList.
Fuente: Propia.

Ejemplo de la implementación de una interfaz en java

En el siguiente ejemplo se usará la interfaces en java. En el diagrama se muestra las clases y sus relaciones:

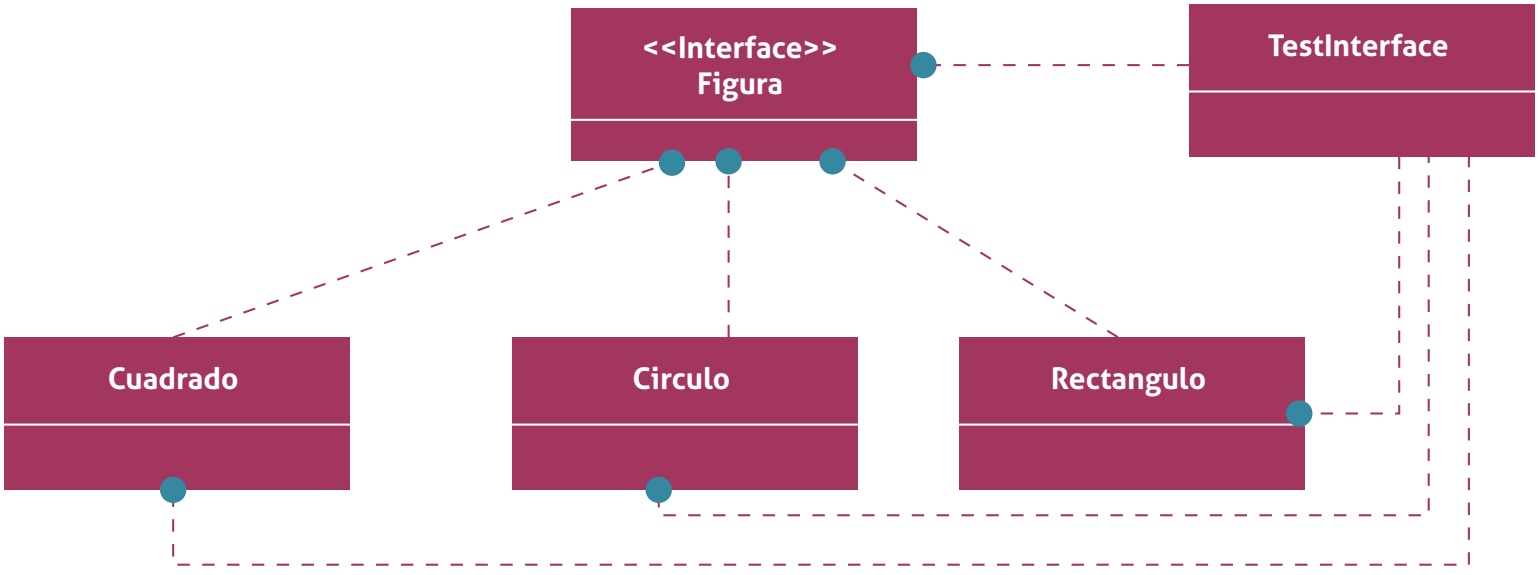


Figura 1.
Fuente: Propia.

Ejercicio 1

public interface Figura{

float PI = 3.1416f;

float área();

}

publicclass Cuadrado implements Figura{ //La clase implementa la interface Figura

privatefloat lado;

public Cuadrado (float lado){this.lado = lado;}

*publicfloat área(){ return lado*lado:}*

}// Cierre de la clase ejemplo

publicclass Circulo implements Figura{ //La clase implementa la interface figura

```
privatefloatdiametro;
public Circulo (float diámetro){ this.diametro = diámetro;}
public float area(){ return (PI * diametro * diámetro/4f);}
} // Cierre de la clase ejemplo

publicclass Circulo implements Figura{ //La clase implementa la interface figura
privatefloatdiametro;
public Circulo (float diámetro){ this.diametro = diámetro;}
public float area(){ return (PI * diametro * diámetro/4f);}
} // Cierre de la clase ejemplo
publicclassRectanguloimplements Figura{ //La clase implementa la interface Figura
privatefloat lado; privatefloat altura;
publicRectangulo (float lado, float altura) {
this.lado = lado;
this.altura = altura;}
public float area(){
returnlado*altura;}
}//Cierre de la clase ejemplo
importjava.util.List; import java.util.ArrayList; //Test ejemplo
public class TestInterface {
        public static void main (String [ ] Args) {
                Figura cuad1 = new Cuadrado (3.5f);
                Figura cuad2 = new Cuadrado (2.2f);
                Figura cuad3 = new Cuadrado (8.9f);
```

```
Figura circ1 = new Circulo (3.5f);

Figura circ2 = new Circulo (4f);

Figura rect1 = new Rectangulo (2.25f, 2.55f);

Figura rect2 = new Rectangulo (12f, 3f);

List <Figura>serieDeFiguras = new ArrayList<Figura> ();

serieDeFiguras.add (cuad1); serieDeFiguras.add (cuad2); serieDeFiguras.add
(cuad3);

serieDeFiguras.add (circ1); serieDeFiguras.add (circ2); serieDeFiguras.add (rect1);
serieDeFiguras.add (rect2);

floatareaTotal = 0;

for (Figuratmp: serieDeFiguras) {     areaTotal = areaTotal + tmp.area(); }

System.out.println ("Tenemos un total de " + serieDeFiguras.size() + " figuras y su
área total es de "  +

areaTotal + " uds cuadradas")

}
```

} //Cierre del main y de la clase

Explicación del código

public interface Figura{

float PI = 3.1416f;

float área();

}

Se crea una interface llamada Figura que define una constante PI y un atributo, ambos de tipo float.

publicclass Cuadrado implements Figura{ //La clase implementa la interface Figura

privatefloat lado;

public Cuadrado (float lado){this.lado = lado;}

*publicfloat área(){ return lado*lado:}*

}// Cierre de la clase ejemplo

Se crea una clase Llamada Cuadrado que implementa la interfaz Figura que se ha creado anteriormente.

En la clase Cuadrado se crea el constructor que tiene el mismo nombre de la clase con un parámetro de tipo float para el lado. Igualmente se crean las clases de Círculo y Rectángulo con los parámetros correspondientes a cada clase. En la clase Círculo se define el atributo de diámetro y en la clase Rectangulo se define los atributos de lado y altura.

La clase*public class TestInterface {*

public static void main (String [ ] Args) {

Figura cuad1 = new Cuadrado (3.5f);

Figura cuad2 = new Cuadrado (2.2f);

Figura cuad3 = new Cuadrado (8.9f);

Figura circ1 = new Circulo (3.5f);

Figura circ2 = new Circulo (4f);

Figura rect1 = new Rectangulo (2.25f, 2.55f);

Figura rect2 = new Rectangulo (12f, 3f);

Permite la creación del objeto de tipo Figura de las clases Cuadrado, Circulo, Rectangulo, con los valores de los parámetros que se han definido en cada clase.

Ejercicio 2

En el siguiente ejercicio, se implementará una clase que permita desarrollar una interface comparable en java.

public class Persona{

publicintdni, edad;

public Persona(int d, int e){

this.dni = d;

```
        this.edad = e;
    }
}
```

Se ha creado una clase "Persona" con dos atributos dni y edad definidos como enteros, luego se implementará la interfaz comparable, y es necesario implementar el método *publicintcompareto*(Persona p); el cual debe devolver un número negativo, 0, o un número entero positivo, como resultado de la comparación del objeto "p" cuando este sea menor, mayor o igual al objeto de referencia.

Posteriormente se incluirá el método *publicintcompareto*(Persona p) en la clase Persona. Si se asume que se tienen 2 personas y se desea ordenarlas o compararlas, se puede hacer con el método compareto. Por su altura o la edad, podríamos preguntarnos cuando una persona sea mayor, menor o igual a otra por algún atributo; para este ejemplo definiremos que una persona es mayor a otra si su edad es mayor, y es menor a otra si su edad es menor, y serán iguales si sus edades son iguales. El criterio de ordenación lo hemos decidido y después de esto lo llevaremos al código. (Rodríguez, 2009).

Se puede aplicar diferentes criterios de ordenación.

Nuestra clase Persona seria la siguiente:

```
public class Persona implements Comparable<Persona>{

publicintdni, edad;

public Persona(int d, int e){

this.dni = d;

this.edad = e;

}

publicintcompareTo(Persona p){

intresultado = 0;

if(this.edad<p.edad){

resultado = -1;

}

else{

if (this.dni<p.dni){

resultado = -1;

}

else if(this.dni>p.dni){

resultado = 1;

}

else {

resultado = 0;}

}

return resultado;

}

}
```

En el ejercicio se ha incluido *"implements Comparable<Persona>"*, esto indica que se va incluir en la clase la interfaz "Comparable" para el parámetro Persona. También se ha incluido el método *"publicintcompareto(Persona p")*, para controlar el caso de que dos personas sean

iguales por la edad, se compare sus dni para definir qué persona es mayor a otra.

El siguiente paso es crear la clase "Programa", que se encargará de crear personas y compararlas para mostrar un mensaje de cuál es la mayor.

```
Public class Programa {

        Public static void main(String [] args){

                Persona p1 = new Persona(123456, 35);

                Persona p2 = new Persona(654321,30);

if (p1.compareto(p2) < 0){ System.out.printl("La persona p1: es menor.");}

elseif (p1.compareto(p2) > 0){System.out.println("La persona p1: es mayor");}

else { System.out.println("La persona p1 es igual a la persona p2");}

}

}
```

Interfaces visuales objetos swing

Los programas en java pueden tener una respuesta al usuario a través de una interfaz gráfica, lo que permite realizar entradas y salidas mediante las interfaces graficas de usuario.

Es importante saber que la implementación de Interface como opción de java consiste en una agrupación de métodos abstractos y propiedades, aspecto que es totalmente diferente a las interfaces graficas de usuario.

Vamos a realizar un ejercicio donde tendremos un formulario que le permita al usuario poder visualizar de forma gráfica la respuesta del programa. (Booch, 1996).

Iniciaremos con el ejercicio de hola mundo.

Programa:

```
importjavax.swing.*;

public class Formulario extends JFrame {

privateJLabel lbl1;

publicFormulario(){
```

```
        setLayout(null);
        lbl1 = new JLabel("HolaMundo");
        lbl1.setBounds(10,20,200,30);
        add(lbl1);
}
public static void main(String [] args){
Formularioformu = new Formulario();
Formu.setBounds(10,10,400,300);
Formu.setVisible(true);
}
}
importjavax.swing.*;
```

El símbolo "*" en la línea import indica que se van a importar todas las clases que conforman el paquete swing, dentro de la aplicación del formulario.

Cuando una aplicación debe importar varias clases del paquete swing, es preferible hacerlo con el asterisco "*", para así incluir todo el paquete en la aplicación.

Para incluirlos de forma individual, hay que definirlos línea por línea:

```
importjavax.swing.JFrame;
importjavax.swing.JLabel;
```

La clase "JFrame" permite encapsular el concepto ventana. Cuando deseemos implementar una aplicación que requiera un formulario, se debe extender de la clase "JFrame" para heredar los objetos y métodos de la clase.

```
publicclasssFormulario extends JFrame{
```

La línea anterior está indicando que la clase "Formulario" hereda las clases y atributos de la clase principal JFrame.

Cuando deseamos mostrar un texto en la pantalla sobre un formulario, requerimos de la clase JLabel, que permite mostrar una etiqueta en pantalla.

Debemos definir un objeto como atributo de la clase JLabel, de la siguiente forma:

```
privateJLabel lbl1;
```

En el constructor de la clase llamamos al método de la clase "JFrame" llamado "setLayout" al cual se le pasa como parámetro un valor "null", con esto le estamos indicando a la clase "JFrame" que se va a utilizar un posicionamiento absoluto para los objetos visuales dentro del "JFrame".

```
public Formulario (){
        setLayout(null);
```

Posteriormente debemos crear el objeto de la clase "JLabel" y pasarle como parámetro al constructor el texto de la etiqueta a mostrar.

```
jbl1= new JLabel("Hola Mundo");
```

Ubicamos al objeto de la clase "JLabel" llamando al método "setBounds", al cual se le definen los parámetros de "columna", "file", "ancho" y "alto" de "JLabel".

Posteriormente debemos añadir al objeto al control del"JFrame", para esto utilizamos el método "add".

```
lbl1 = new JLabel("HolaMundo");

jbl1.setBounds(10,20,200,30);

add(jbl1);

}
```

Finalmente hacemos uso del método "main", para crear el objeto de la clase "Formulario", llamando al método "setBounds" con los parámetros indicados y una última instrucción para hacer visible el texto en el "JFrame", con "setVisible".

```
public static main(String[] args){

        Formularioformu = new Formulario();

        Formu.setBounds(10,10,400,300);

        Formu.setVisible(true);

}
```

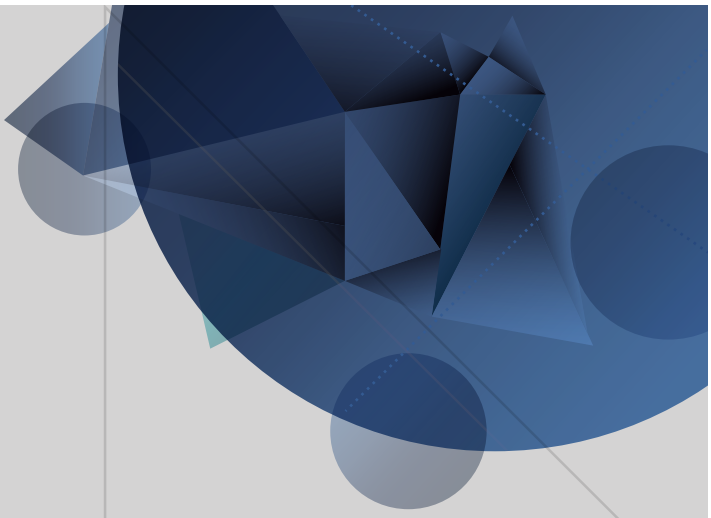
Al ejecutar el programa obtendremos un resultado igual a este.

Imagen 1.
Fuente: http://danha.tistory.com/990

4

Unidad 4

Polimorfismo en java



Modelos de programación I

Autor: Gustavo Enrique Tabarez Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

La programación en java, permite tener diferentes formas de reutilizar el código logrando con esto tener una programación mucho más eficiente y cumpliendo con las buenas prácticas de programación.

El polimorfismo permite lograr que el código en la programación este mucho más estructurado hacia las diferentes respuestas que pueden adoptar los métodos cuando reciben diferentes entradas y pueden dar respuesta de acuerdo a sus comportamientos.

U4 Metodología

Para cada unidad el estudiante cuenta con un documento, en el que se describe las actividades a realizar para la semana; dicho documento se denomina *guía de actividades*. Asimismo le informa las tareas calificables e incluye los objetivos de aprendizaje. Antes de iniciar usted debe leerlo muy atentamente.

De otro lado, cada semana cuenta con lecturas complementaras, recursos para el aprendizaje, actividades de repaso y videoconferencias; que le permitirán aclarar dudas y/o profundizar en la temática semanal.

La actividad evaluativa le permitirá al estudiante medir su nivel de conocimiento frente al tema y poder profundizar en aquellos tópicos en que se encuentre más débil.

Polimorfismo y reutilización

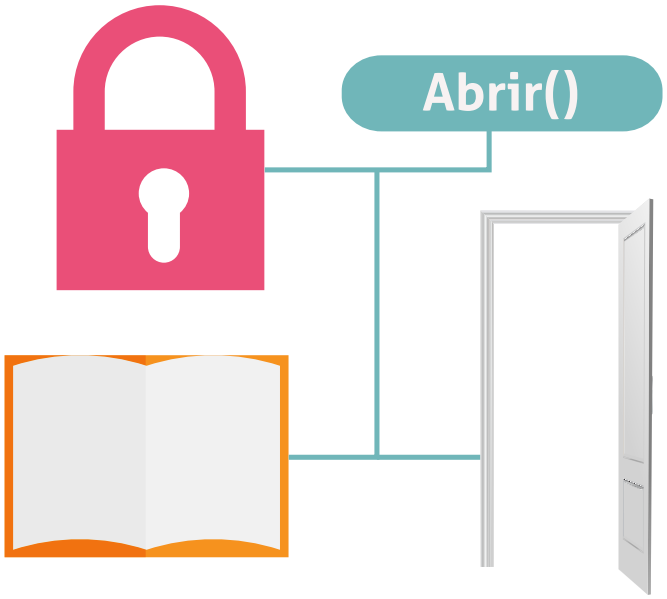


Figura 1. Estado de objetos.
Fuente: Propia.

Los objetos mostrados en la figura, tienen una propiedad en común a la cual denominaremos "estado", el cual será modificado por los métodos de Abrir() y Cerrar().El método es igual para los objetos, su comportamiento o respuesta a la petición de cada objeto será diferente. Si analizamos el objeto candado y el objeto libro ambos comparten el método abrir pero la acción será diferente a cada uno de ellos, ya que no es lo mismo abrir un candado que un libro, aunque compartan el mismo objeto.

La propiedad de polimorfismo permite que métodos que tienen el mismo nombre dentro de una aplicación y funcionalidad, den respuestas de forma diferente cuando son invocados por objetos, y su comportamiento dependerá de la naturaleza del objeto dentro del cual actúan.

El Polimorfismo, desde la programación orientada a objetos, permite que varias clases de diferentes tipos, puedan estar referenciadas por una sola variable.

Veamos un ejemplo, una instancia de la clase Figura podrá referenciar a atributos y métodos definidos en la clase "Circulo" y "Rectangulo". Veamos un ejemplo tomado de la página programación orientada a objetos con java. (Pizarro, 2010):

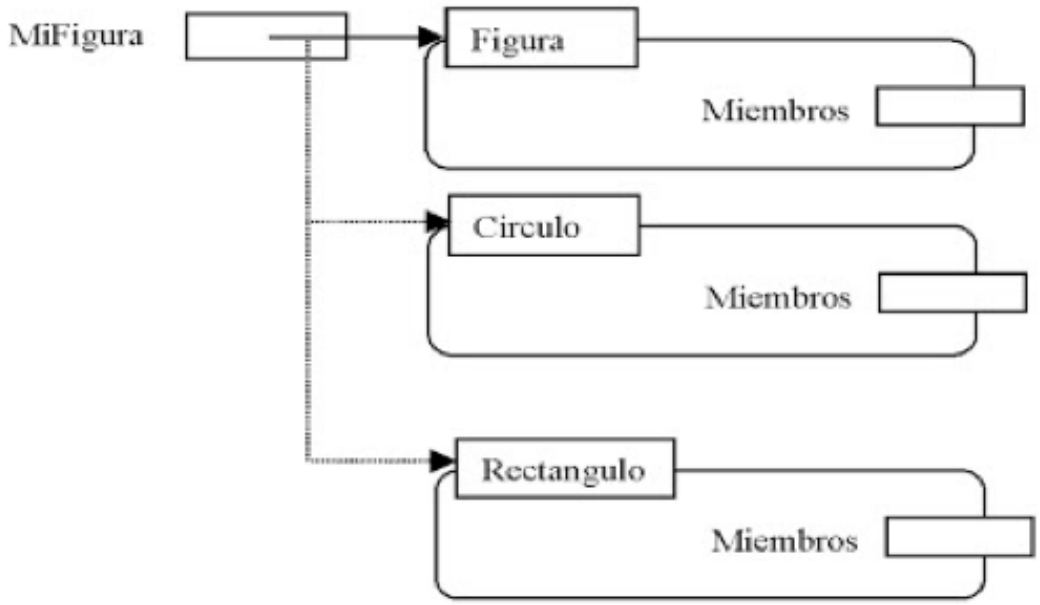


Figura 2. Figura.
Fuente: propia.

Tomemos como ejemplo que las clases "Circulo" y "Rectangulo" implementan un método llamado "Perimetro" encargado de calcular el perímetro de la figura.

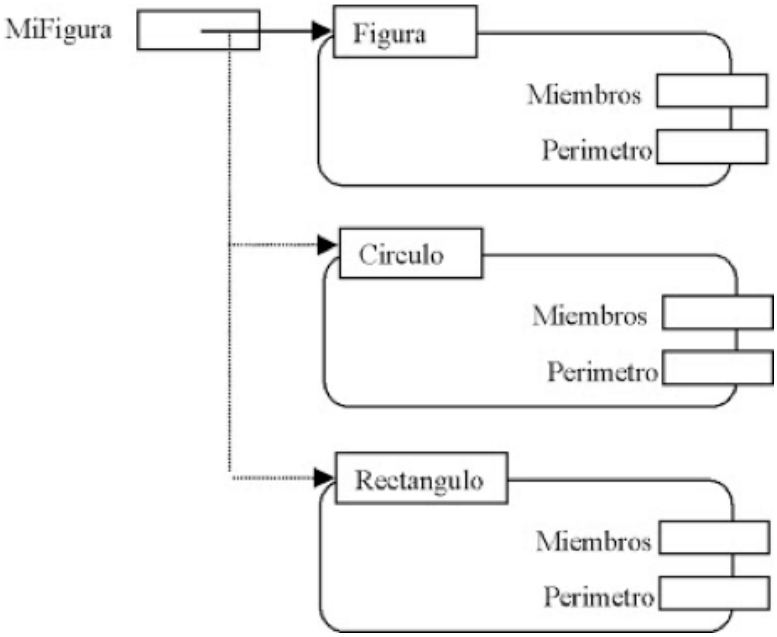


Figura 3 Figura.
Fuente: propia.

La referencia que se hace a la clase "MiFigura", podrá llamar el método "Perimetro" de la clase "Circulo" o al método "Perimetro" de la clase "Rectangulo", todo está dependiendo, en el momento de ejecución, si está referenciando a una instancia de la clase "Circulo" o si está referenciando a una instancia de la clase "Rectangulo". (Pizarro, 2010). Pendiente

Es muy importante tener en cuenta que la determinación del método "Perimetro" que se desea invocar, ya sea de la clase "Circulo" o el de la clase "Rectangulo", este no se realiza en la compilación sino al momento de la ejecución, característica muy importante del polimorfismo.

a continuación se presenta un cuadro en el que se ilustra en qué consiste el polimorfismo y la diferencia que tiene una clase con herencia y una clase abstracta con polimorfismo.

CLASE PADRE Métodos: **Escribir** (escribe un texto en un archivo) **leer**((lee un archivo escrito)	Clase hija: Métodos: **ordenar** (ordena una serie de caracteres en un texto), Escribir (escribe un texto en un archivo).
	Clase hija: Métodos: **desordenar** (desordena una serie de caracteres en un texto), Escribir (Escribe un texto en un archivo).

En el cuadro se representa una clase Padre compartiendo métodos totalmente iguales a sus subclases, y las subclases tienen sus propios métodos, pero las subclases pueden hacer uso sin ninguna restricción de los métodos de la clase Padre.

Polimorfismo

CLASE PADRE Métodos: **Escribir,Leer**	Subclase 1: Método: **Escribir** (Escribir números en un archivo), **ordenar** (ordena una serie de caracteres en un texto).
	Subclase 2: Método **Escribir** (Escribe todas las pulsaciones del teclado), **desordenar** (desordena una serie de caracteres en un texto).

En este ejemplo solo tenemos los nombres de los métodos y en las subclases se definen las funciones que toman los métodos, el método escribir hace una cosa totalmente diferente a las subclases, pero sigue usando el método escribir de la clase abstracta.

Veamos en el siguiente código la implementación del polimorfismo:

```
/*
*
* @author Gustavo – Clase Abstracta
*/
abstractclassFiguraPadre {
//solo se definen los métodos
abstract void area();
abstract void perimetro();
}
classCuadrado extends FiguraPadre{
void area(){
        System.out.println("Método para calcular el área de un cuadrado");
    }
voidperimetro(){
        System.out.println("método para calcular el perímetro de un cuadrado");
   }
}
classCirculo extends FiguraPadre {
void area() {
        System.out.println("Método para calcular el área de un circulo");
}
void perímetro(){
        System.out.println("Método para calcular el perímetro de un circulo");
```

```
}
}
public class ClasePolimorfismo{
public static void main(String[] args) {
        FiguraPadre fig1 = new Circulo();
        fig1.area();
        fig1.perimetro();
        fig1 = new Cuadrado();
        fig1.area();
        fig1.perimetro();
}
}
```

En el código podemos observar que está compuesto por varias clases llamadas

classFiguraPadreque solo define los métodosabstractvoidarea() y

abstractvoidperimetro().

class Cuadrado extendsFiguraPadrehereda de la clase FiguraPadre

class Circulo extendsFiguraPadrehereda de la clase FiguraPadre

publicclassClasePolimorfismose encarga de instanciar los objetos de cada clase

Tipos de polimorfismo

Sobrecarga: (Overloading, Polimorfismo ad-hoc) un solo método el cual tiene un nombre definido y muchas respuestas distintas.

Las funciones sobrecargadas se diferencian en tiempo de compilación por estar compuestas de varios tipos de entrada y/o salida.

Sobreescritura (Overriding, Polimorfismo de inclusión) Es el tipo de sobrecarga especial que se da en las relaciones de herencia.

La funcionalidad es el mismo reemplazo del método del padre, aunque los métodos se encuentran en diferentes clases, se encuentran relacionados mediante la herencia.

Variables polimórficas (Polimorfismo de asignación) Tipos de variables que se declaran de un tipo, pero en realidad referencian un valor de un tipo distinto.

Genericidad (plantillas o templetes) Son elementos de propósito general (Clases o métodos) y son definidas para situaciones específicas. (Cristina Cachero, 2014). Pendiente

Ejemplos:

Sobrecarga:

Factura::imprimir()

Factura::imprimir(intnumCopias)

ListaCompra::imprimir()

Sobreescritura

Cuenta::abonarInteres()

CuentaJoven::abonarInteres()

Valiables polimórficas

*Cuenta *pc = new CuentaJoven();*

Genericidad

Lista<Cliente>

Lista<Articulo>

Lista<Alumno>

Lista<Habitación>

La sobrecarga

La sobre carga de métodos consiste a que un mismo nombre puede representar a varios métodos distintos con diferentes tipo y número de parámetros, todos manejados dentro en una sola clase. En la programación orientada a objetos, el tema de sobrecarga se refiere a tener varios métodos con el mismo nombre pero con una funcionalidad diferente. El caso más utilizado son los constructores que permiten tener el mismo nombre de la clase, pero su función es diferente, pues este se encarga de inicializar atributos o métodos.

La forma de diferenciar los métodos sobrecargados es por su número de parámetros, analizando el tipo, la cantidad o el orden de estos. Para comprender más en detalle el tema de sobrecarga se analiza a continuación un ejemplo:

```
public class Producto{

        privatefloarprecio;

        publicvoidsetPrecio(){

                precio = 3.50;

        }

publicvoidsetPrecio (floatnuevoPrecio){

        precio = nuevoPrecio;

}

publicvoidsetPrecio(float valor, intporcentajedescuento){

precio = valor + (valor * porcentajedescuento);

 }

}
```

Como se puede observar en el ejemplo, se tiene una clase llamada "Producto" con tres métodos, donde el primero no tiene parámetros, el segundo tiene un parámetro de tipo float llamado "nuevoPrecio", y el tercero tiene dos parámetros uno de tipo entero y otro de tipo float.

Ejemplo de sobreescritura: esta se aplica a los métodos, donde los métodos de la clase padre se redefinen en las subclases. En la relación de herencia de la clase Figura tiene definido los métodos "calcularArea" y "calcularPerimetro", las subclases "Cuadrado", "Circulo", "Triangulo" y "Rectangulo" que heredan de la clase "Figura", estas redefinen estos métodos ya que sus cálculos son diferentes.

Veamos el ejemplo:

```
public class Figura{
protecteddoubléárea;
protecteddoubléperímetro;
publicFigura(){
this.area = 0;
this.perimetro= 0;
}
public double getArea(){
returnárea;
}
public double getPerimetro() {
      return perimetro;
   }
   public void calcularArea(){}
   public void calcularPerimetro(){}
}
public class Circulo extends Figura {
   private double radio;
   public Circulo() {
      super();
   }
   public double getRadio() {
       return radio;
```

```
    }
   public void setRadio(double radio) {
this.radio = radio;
    }
   public void calcularArea() {
this.area = Math.PI*Math.pow(this.radio,2.0);
    }
   public void calcularPerimetro() {
this.perimetro = 2*Math.PI*this.radio;
 }
}
```

Explicación de código:

```
public class Figura{
protecteddoublé area;
protecteddoubléperímetro;
publicFigura(){
this.area = 0;
this.perimetro= 0;
}
```

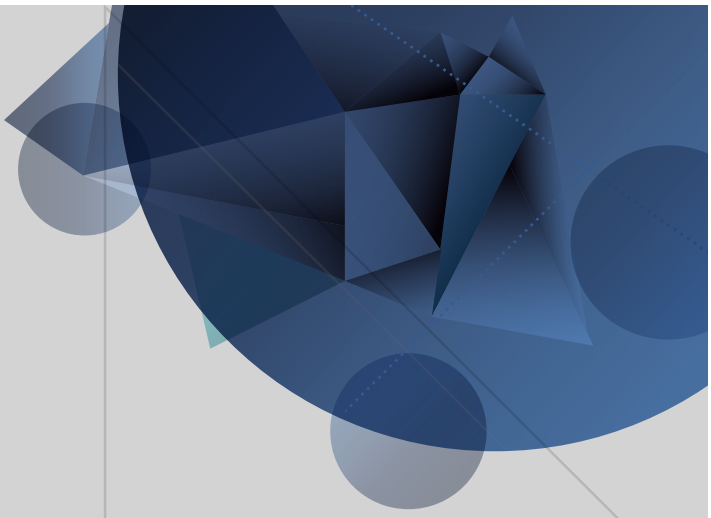
Se crea una clase llamada "Figura" que tiene los atributos de "área" y "perímetro" y se define el constructor Figura que se encarga de hacer una sobrecarga sobre los atributos inicializándolos en cero "0".

Se crean los métodos *getArea(), getPerimetro(), calcularArea(), publicclass Circulo extends* Figura que tendrán los comportamientos diferentes de los métodos, y pueden ser llamados desde diferentes clases.

4

Unidad 4

Interfaces graficas jframe



Modelos de programación I

Autor: Gustavo Enrique Tabarez Parra

AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO

Introducción

El desarrollo de aplicaciones en java con interfaces gráficas, permite a los usuarios tener una mayor interactividad con una aplicación. Las interfaces gráficas dan a las aplicaciones presentaciones amigables que facilitan el intercambio de la información entre la aplicación y el usuario.

La generación de código automático que se produce al crear interfaces gráficas, le permiten al desarrollador agilizar y controlar de forma eficiente la producción de interfaces GUI, y posteriormente analizar el código generado comprendiendo la sintaxis para complementar el desarrollo, personalizando la aplicación y logrando los resultados esperados.

U4 Metodología

Es importante llevar a cabo el desarrollo de la semana de una forma secuencial, para tener claridad en el tema; se recomienda no saltar entre páginas, ya que el contenido está enfocado al seguimiento paso a paso de una actividad.

Es importante apoyar el tema con lecturas recomendadas en la bibliografía y en los recursos de las lecturas complementarias.

JFC y Swing.

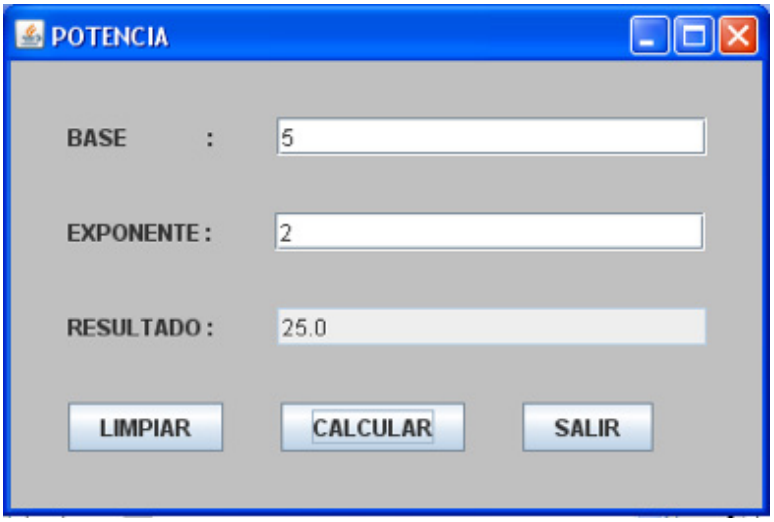


Figura 1. Representación de la estructura de una aplicación gráfica con la clase Swing.
Fuente: http://javadhc.blogspot.com/2012/10/ventana-de-operacion-de-potencias-con.html

JFC significa Java TM FoundationClasses, conforma un grupo de características y facilitan la construcción de interfaces gráficas de usuario (GUIS). Una interfaz está integrada por varios métodos declarados que son implementados mediante clases.

Las interfaces pueden estar implementadas por varias clases, métodos y constantes.

Los componentes Swing: incluyen desde botones hasta splitpanes o tablas.

Soporte de aspectos y comportamiento conectable. Ofrece la facilidad a cualquier componente Swing de tener una amplia colección de elementos que permiten dar forma a diferentes aspectos y comportamientos propios de java o un entorno similar a Windows.

El diseño de una interfaz cumpliéndose deben cumplir los siguientes pasos:

- Decidir la estructura de la interfaz: qué elementos gráficos se van a utilizar, y cómo se van a relacionar.

- Definir la disposición de los componentes, de los cuales existen dos tipos: contenedores y componentes atómicos.
- Los contenedores sirven para organizar los componentes contenidos en los mismos y la organización se define como disposición (layout).
- Definir el comportamiento de la interfaz: (Gestión de eventos).
- Los controles: Permiten interactuar con el usuario ante reacciones de eventos, programando las respuestas que generan los elementos al momento de ser invocados.
- Es conveniente mantener la interfaz de la lógica lo más independiente posible. (Mestras, 2004).

Estructura de una aplicación gráfica con Swing.

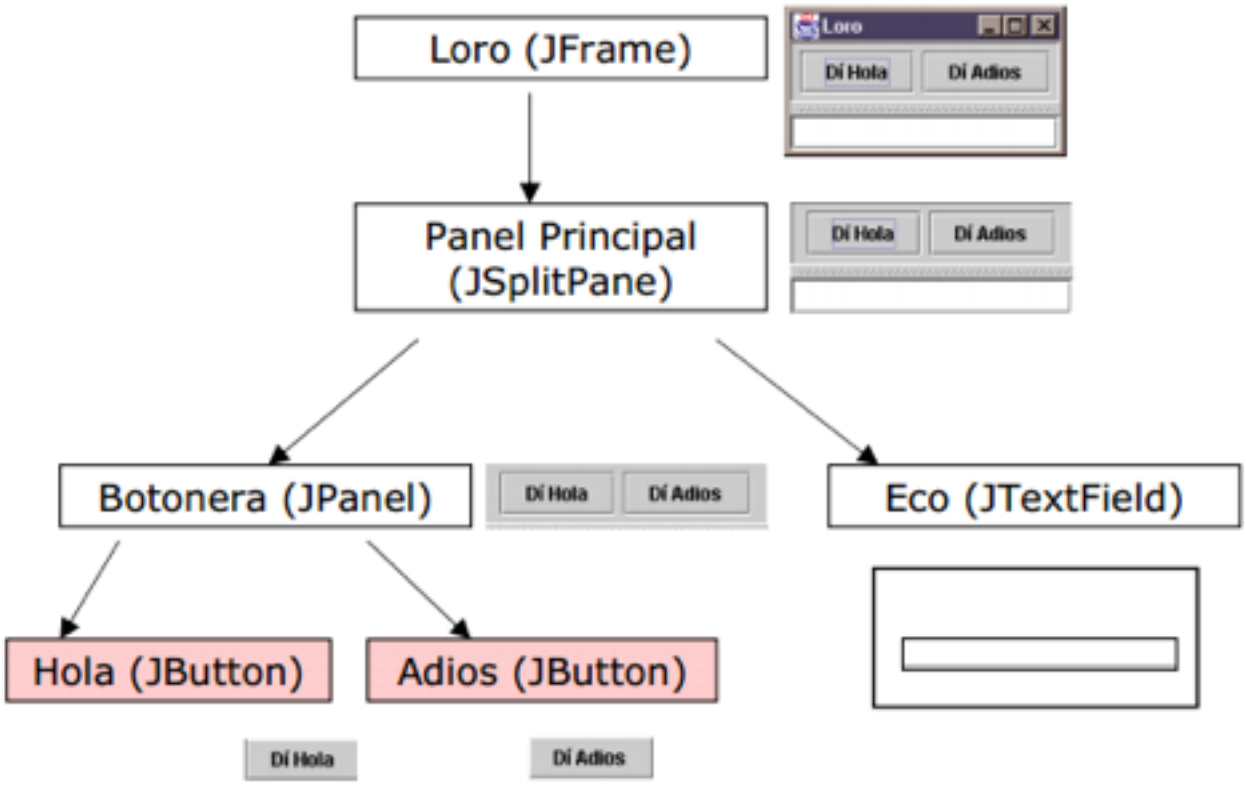


Figura 2 Representación de la estructura deuna aplicación gráfica con la clase Swing.
Fuente: https://www.fdi.ucm.es/profesor/jpavon/poo/2.11.GUIEjercicioSwing-HolaAdios.pdf

Ejemplo de código para la implementación sencilla de una interfaz gráfica.

```
importjavax.swing.*; // Se incluye la clase Swing en la aplicación.
publicclass Loro extendsJFrame{ // creación de la clase Loro extendiendo de JFrame.
        privateJTextField eco;  // Objeto para la entrada de datos
        public Loro() {  //método de la clase
                setTitle("INTERFAZ GRAFICA");
                JComponent botonera = creaBotonera();
                JComponent eco = creaECO();
                JSplitpane (JSplitPane.VERTICAL_SPLIT, botonera, eco); // se crea una interfaz con botones.
                getContentPane().add(panelPrincipal); //se adiciona el componente a la interfaz.
                pack(); //permite redimensionar toda la interfaz gráfica en la ventana
                setVisible(true); // se hace visible la ventana
}
privateJComponentcreaBotonera(){
                JPanelbotonera = new JPanel();
                JButtonhola = new JButton("Hola");
                JButton adiós = new JButton("Adios"); // creación de botones
                botonera.add(hola);
                botonera.add(adiós);
                returnbotonera;
        }
PrivatejComponentcreaEco(){
 Eco = new JTextFlied("Pulsa el botón"); //Se crea el campo de texto donde poner el eco.
```

```
return eco;

}

public static void main (String [] args){

Loro loro = new Loro();

  }

}
```

A continuación se verá un ejemplo donde se explica cómo crear una interfaz en java con la clase "JFrame".

Sigue el paso a pasohasta terminar el ejercicio.

Para este ejercicio utilizaremos el IDE (Interface DevelopmentEnvironment) Netbeans, que es una aplicación diseñada para el desarrollo de aplicaciones básicas y avanzadas en java, utilizando un ambiente gráfico e implementando la clase "Swing".

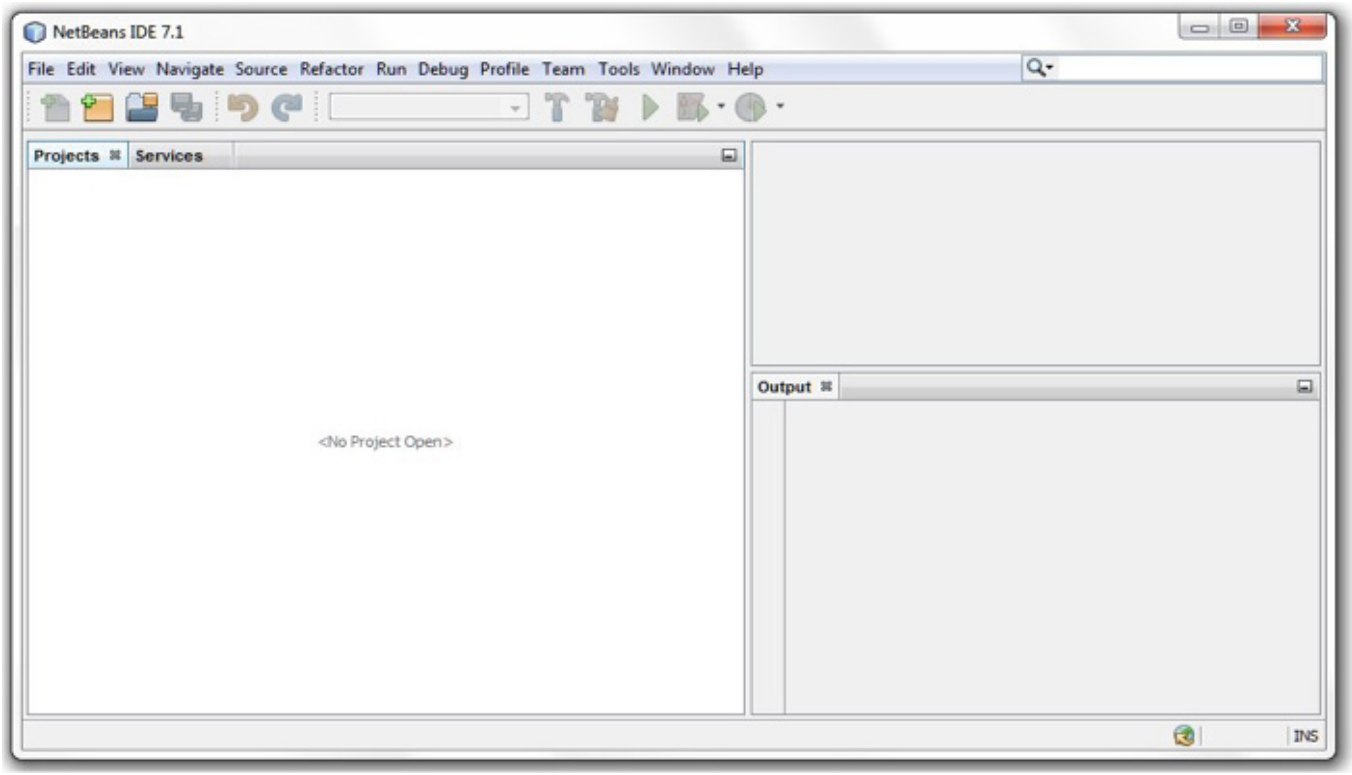


Figura 3. Interfaz de desarrollo IDE Netbeans.
Fuente: propia.

Durante el desarrollo del ejercicio se realizará una calculadora haciendo uso de la interface gráfica.

Crear una calculadora que permita sumar dos números, utilizando una interfaz gráfica generada en Netbeans.

Se inicia un nuevo proyecto llamado sumadora y desactivar la opción de créate mainclass y set as Main Project.

Posteriormente, en el sourcepackageDefault se crea un new "JFrameForm"

Lo anterior debe realizarle en el siguiente orden.

Click en el icono **Netbeans**

Clic en **File**

Click en **New Project**

Seleccione**Java - Java Application**

Click en **Next**

Desactivarlasopciones**Create Main Class ySet as Main Project**

Escriba como nombre de proyecto:**sumadora**

Click en boton Finish

Click en **Source Package - Default - New - Jframe Form**

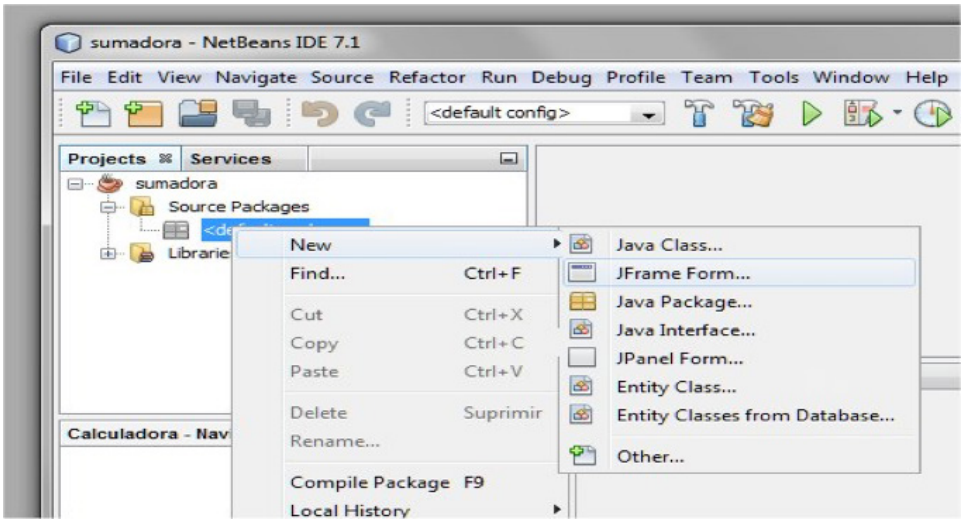


Figura 4. Creación de la clase JFrameForm.
Fuente: propia.

ClassName: sumadora

Package: sumadora

ClickFinish

Se seleccionan de la paleta de herramientas, los objetos de JLabel1 y TextField1, se escriben las propiedades de:

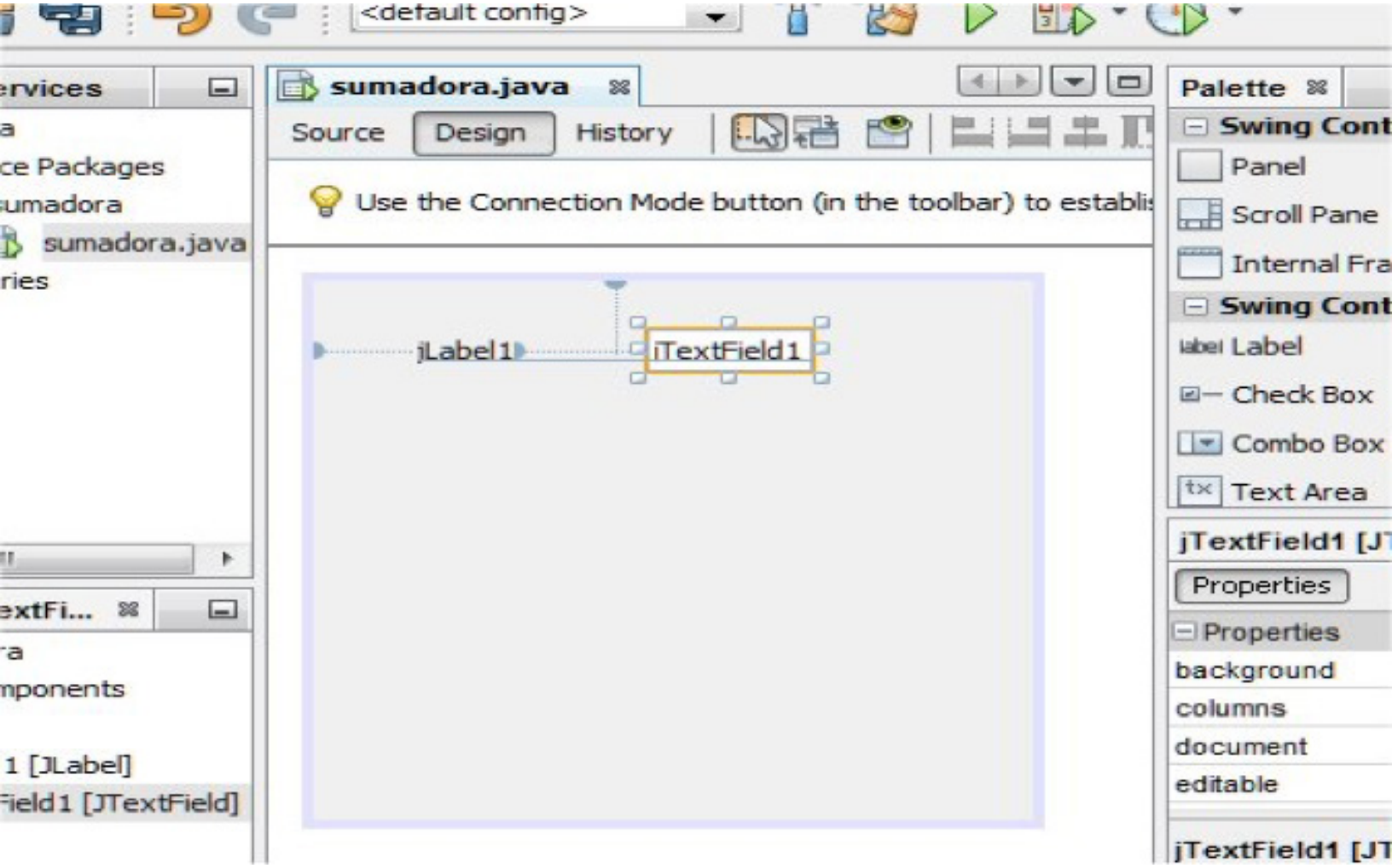


Figura 5 Creamos la interfaz gráfica.
Fuente: propia.

Se agregan los controles JLabel y TextField.

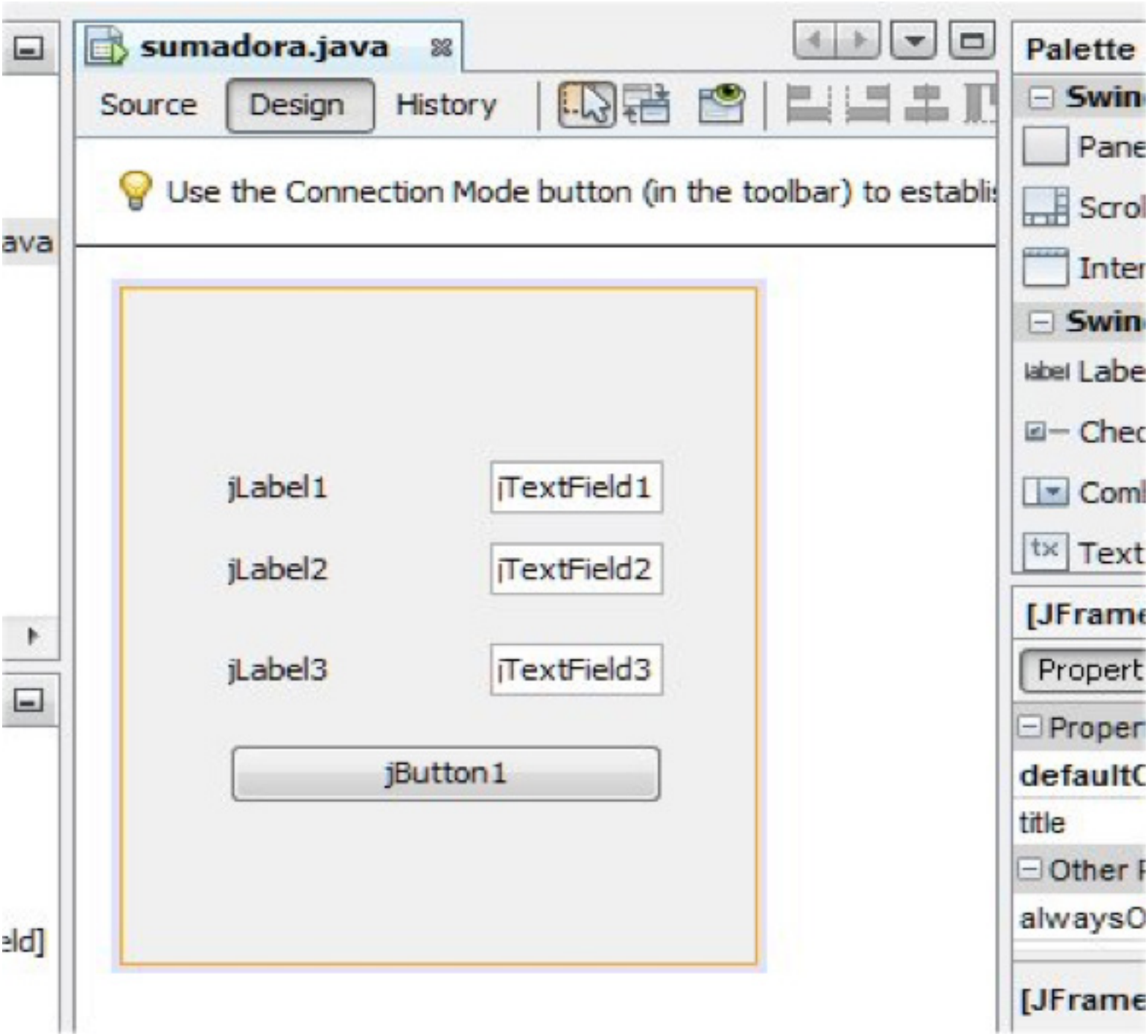


Figura 6. Interfaz de la aplicación.
Fuente: propia.

Se cambian los textos de los campos usando el botón derecho del mouse.

Click en el objeto y luego click derecho del mouse sobre Edit Text.

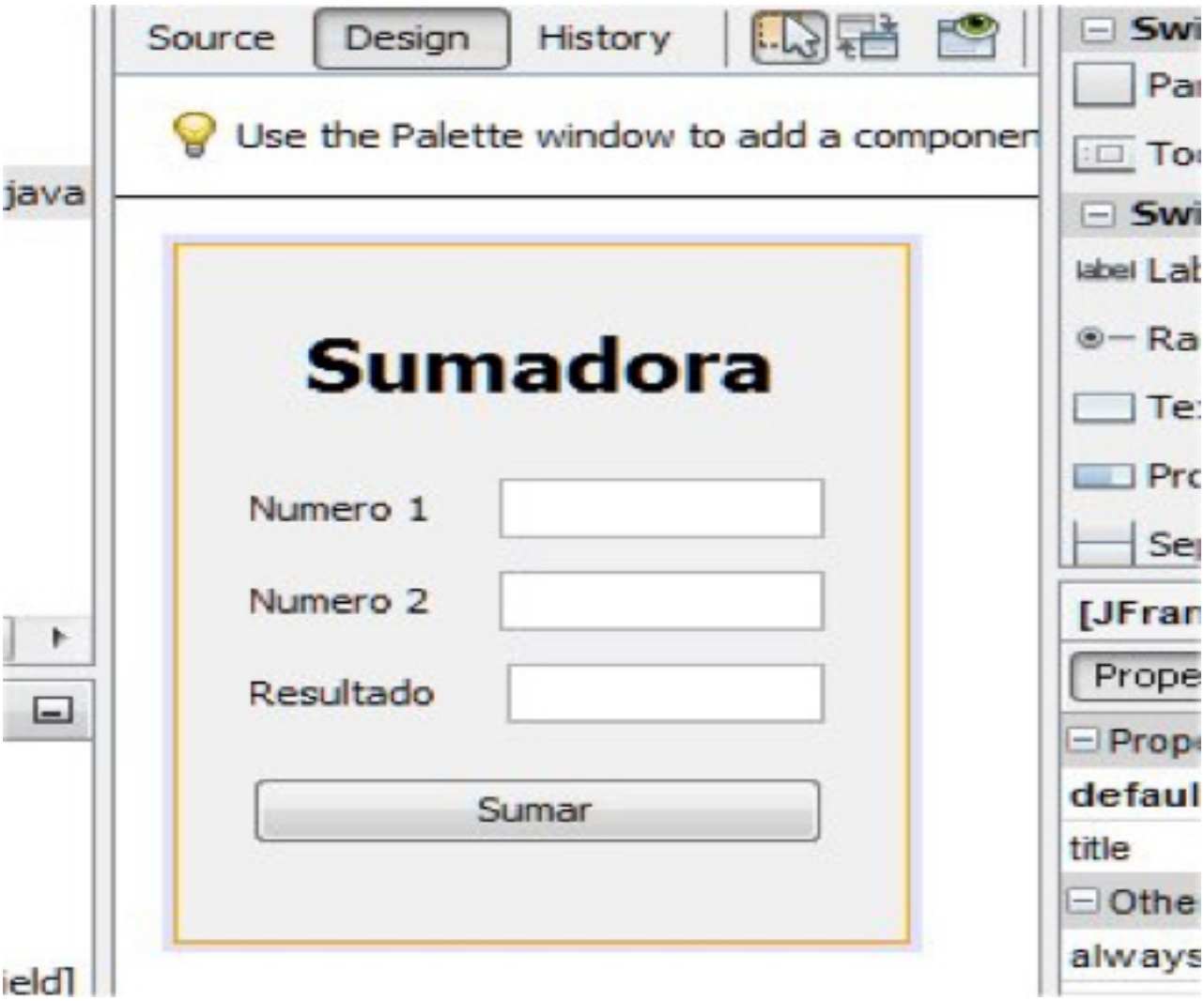


Figura 7. Interfaz de la calculadora terminada.
Fuente: propia.

Se inicia la codificación de la aplicación con el botón Sumar

Doble clic en el botón Sumar

Aparece este código que se va a modificar:

private void jButton1ActionPerformed(java.awt.event.ActionEventevt) {

}

// TODO add your handling code here:

Donde dice // TODO addyourhandlingcodehere: escriba el siguiente código

double nro1, nro2, suma;

nro1 = Double.parseDouble(jTextField1.getText());

nro2 = Double.parseDouble(jTextField2.getText());

suma = nro1 + nro2;

jTextField3.setText(String.valueOf(suma));

Click en la pestaña superior **Design**

Click derecho en el formulario y luego click en **Properties**

Escriba al frente de **Title:** Sumadora

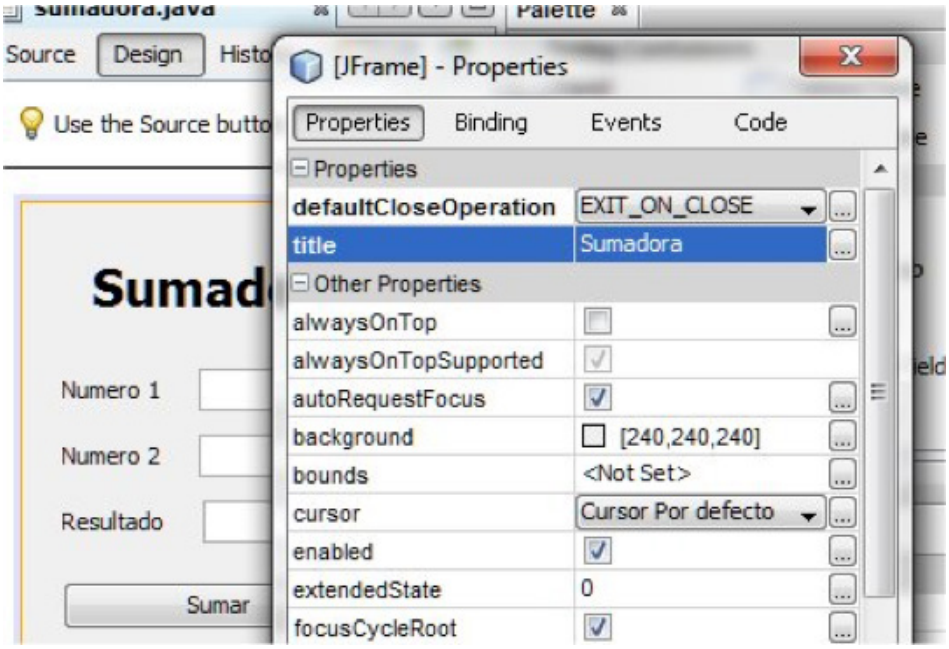


Figura 8. Propiedades del formulario.
Fuente: propia.

Click en Run y luego Click en Set Main Project, después en Sumadora.

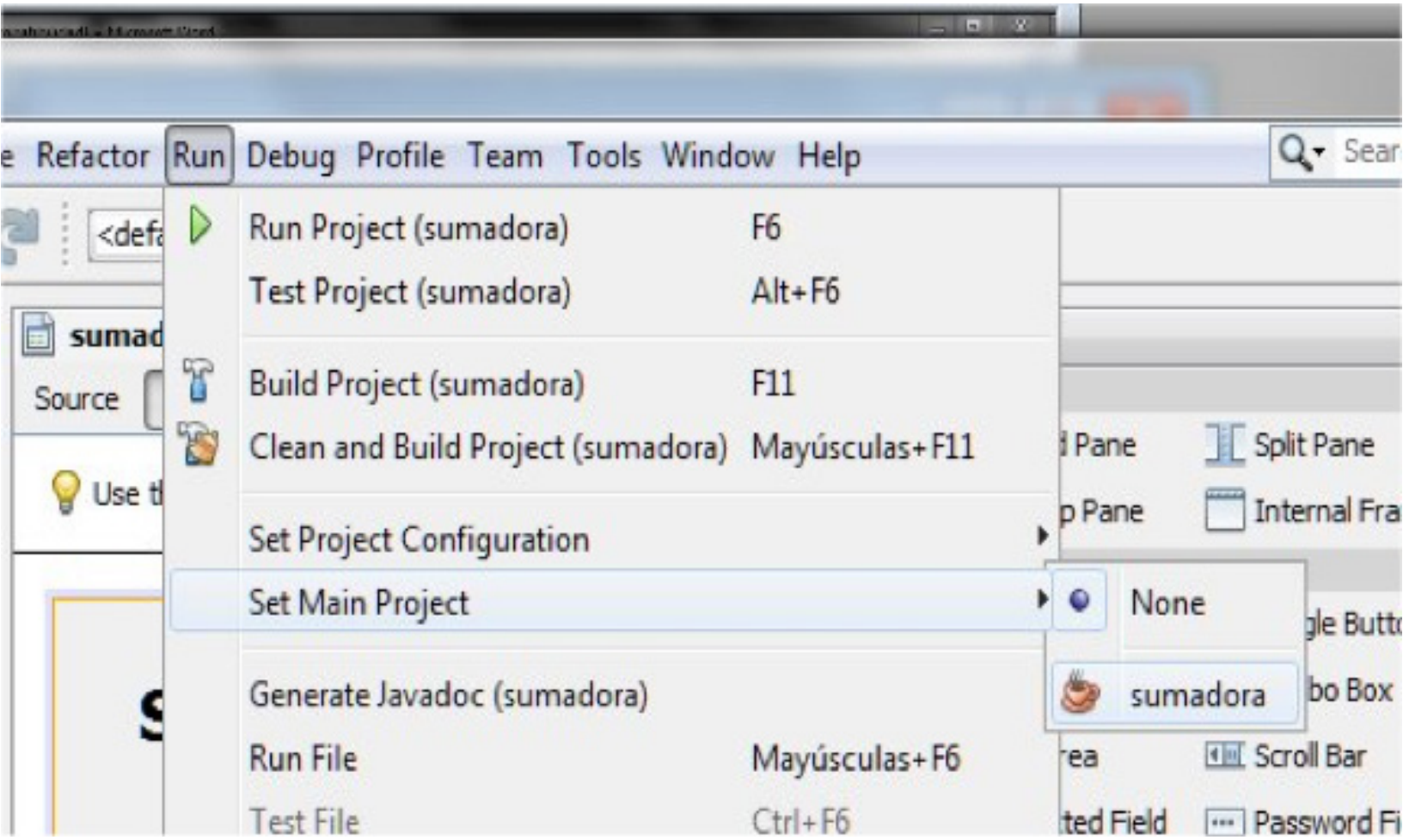


Figura 9. Seleccionar el proyecto como principal.
Fuente: propia.

Click en el botón verde de ejecutar o presione F6

Click en Ok para seleccionar la clase principal (SelecttheMainClass)

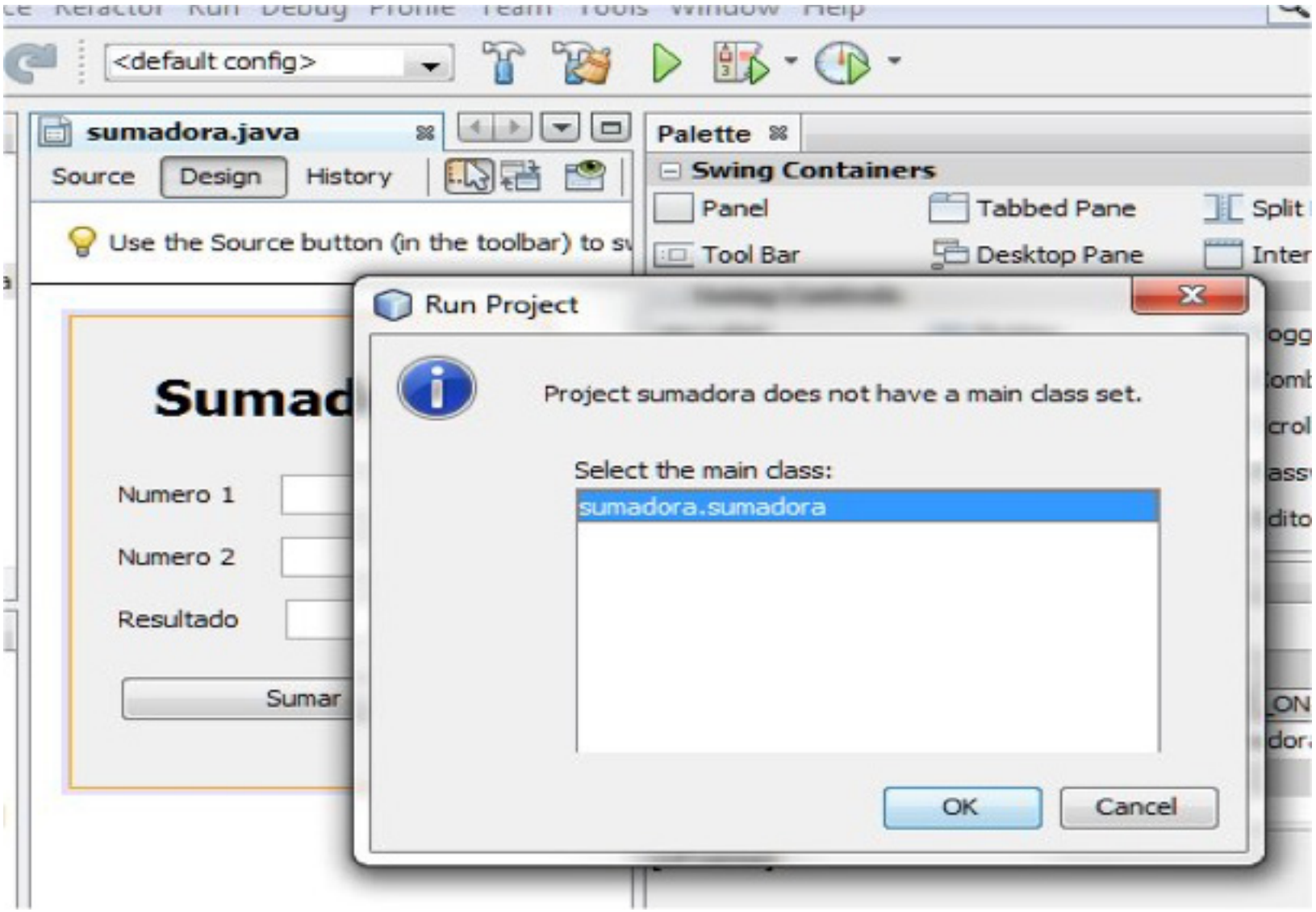


Figura 10. Seleccionar la clase main del proyecto.
Fuente: propia.

Escriba unos valores en los números 1 y 2 y click en el botón Sumar.

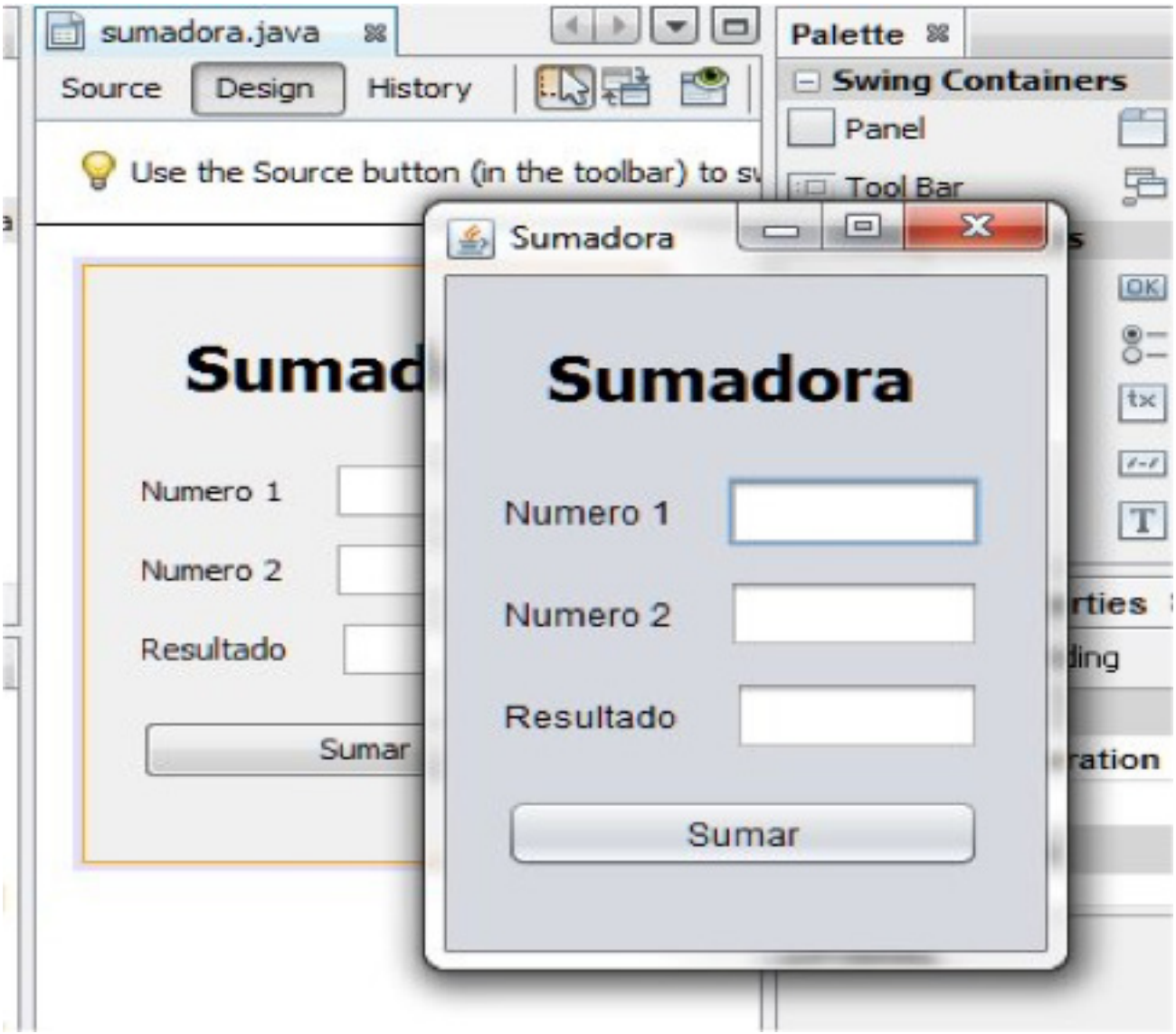
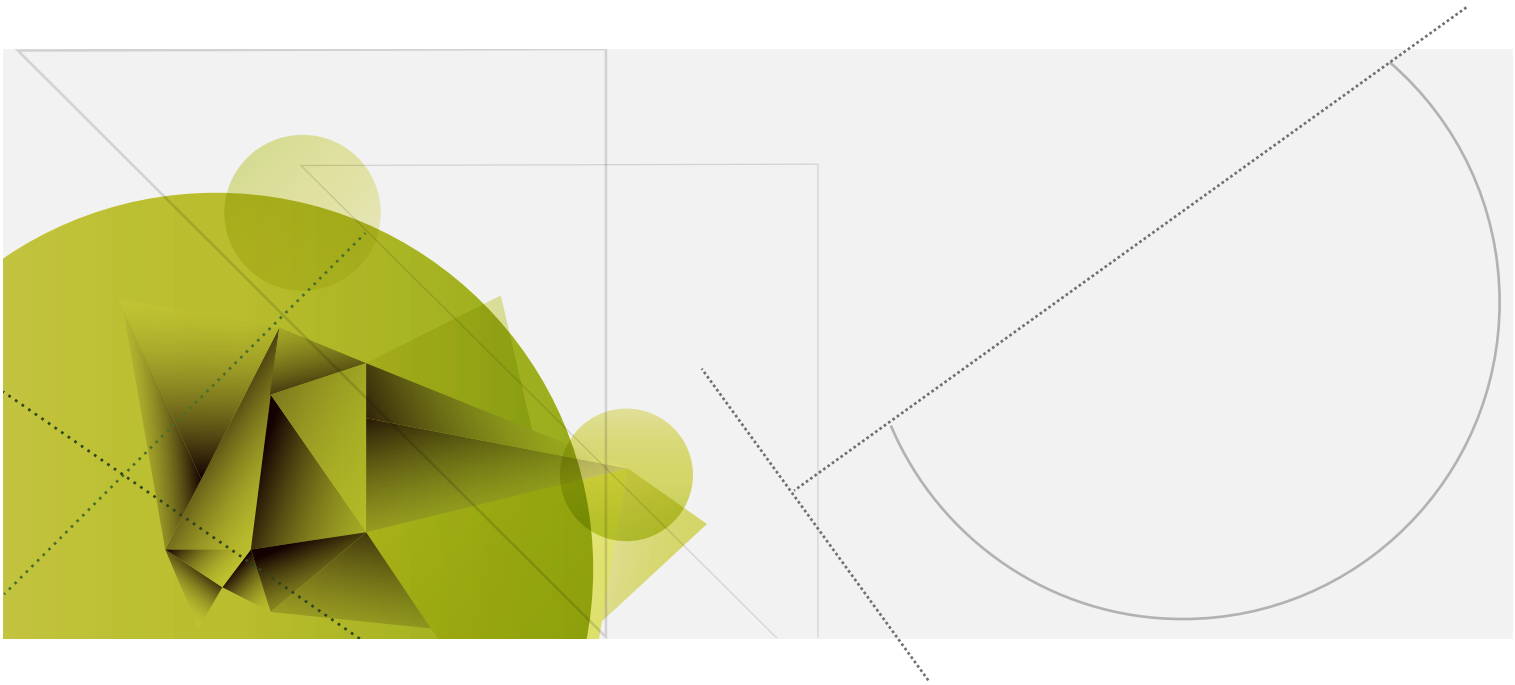


Figura 11. Ejecución de la aplicación terminada.
Fuente: propia.

Bibliografia

- Azpitarte, R. (2009). Introducción a la programación orientada a objetos con Java. España: Departamento de sistemas informáticos y computación. Universidad Politécnica de Valencia.
- Cachero, C. & Ponce de León, P. (2014). Polimorfismo. Recuperado de http://rua.ua.es/dspace/bitstream/10045/4013/1/4-Polimorfismo.pdf
- González, G. (2014). Ciberaula. Recuperadode http://www.ciberaula.com/articulo/lista_programas_que_usan_poo/
- Mateu, L. (1996). Encapsulamiento de clases. Recuperado de http://dcc.uchile.cl/~lmateu/Java/Apuntes/encapsu.htm
- Onrubia, J. & Rochera, M. & Barberà, E. (2001). La enseñanza y el aprendizaje de las matemáticas: una perspectiva psicológica. Madrid: editorial Alianza.
- Rodríguez, A. (2006). Concepto de interfase y herencia múltiple en Java. Recuperado de http://aprenderaprogramar.com/index.php?option=com_attachments&task=download&id=604
- Booch, G. (1996). Análisis y diseño orientado a objetos. Mexico: Addison Wesley Longman.
- Figueroa, K. (2012). Programación en Java. Recuperado de http://todoenjava.blogspot.com/2012/01/herencia.html
- Granada, U. (2014). Ciencias de la computación e inteligencia artificial. Recuperado de http://decsai.ugr.es/~jfv/ed1/c/cdrom/cap7/cap71.htm
- Universidad tecnológica nacional. (2006). Modularidad. Recuperadode http://labsys.frc.utn.edu.ar/aed-2005/Unidad%20II/Apunte/U2%20-%20Modularizaci%F3n.pdf
- Pizarro, J. (2010). Programación orientada a objetos con Java. Recuperado de http://java-jcp.blogspot.com/2010/04/unidad-6-polimorfismo-y-reutilizacion.html

Esta obra se terminó de editar en el mes de noviembre
Tipografá Myriad Pro 12 puntos
Bogotá D.C,-Colombia.



AREANDINA
Fundación Universitaria del Área Andina

MIEMBRO DE LA RED
ILUMNO