Yo@prendo informáticos UNIDOS;
{PROYECTO ACADÉMICO}

Para la carrera:
PTB - INFORMÁTICA

Tipo de Módulo
Profesional

Diseño gráfico: mtra. Alejandra del Ángel López

Módulo:

Programación Orientada a Objetos

Cuarto Semestre

Presenta: Marilú Rivas García
Maestra del Plantel Conalep - Xalapa

conalep
Manuel Rivera Cambas

Contenido del *cuadernillo*

- Palabras del docente
- Actividades
- Conclusión

Yo@prendo informáticos UNIDOS;

{PROYECTO ACADÉMICO}

•modalidad **cuadernillos**•

Academia de Informática

Conalep - Xalapa

"Querido alumno(a):"

El presente material, será una guía de apoyo que te servirá para que tu puedas, desde casa, contar con un recurso escolar, que te permita adquirir y reforzar conocimientos que vemos dentro de nuestro salón de clases.

"Espero te sea de gran apoyo".

Como tu maestro, estoy convencido, que tus ganas y deseos de cumplir con tus tareas y actividades, dentro de Conalep, son las que te impulsan a ser una persona responsable y mejor cada día.

Atentamente:

Tu maestro de informática en Conalep

Yo@prendo informáticos UNIDOS;

{PROYECTO ACADÉMICO}

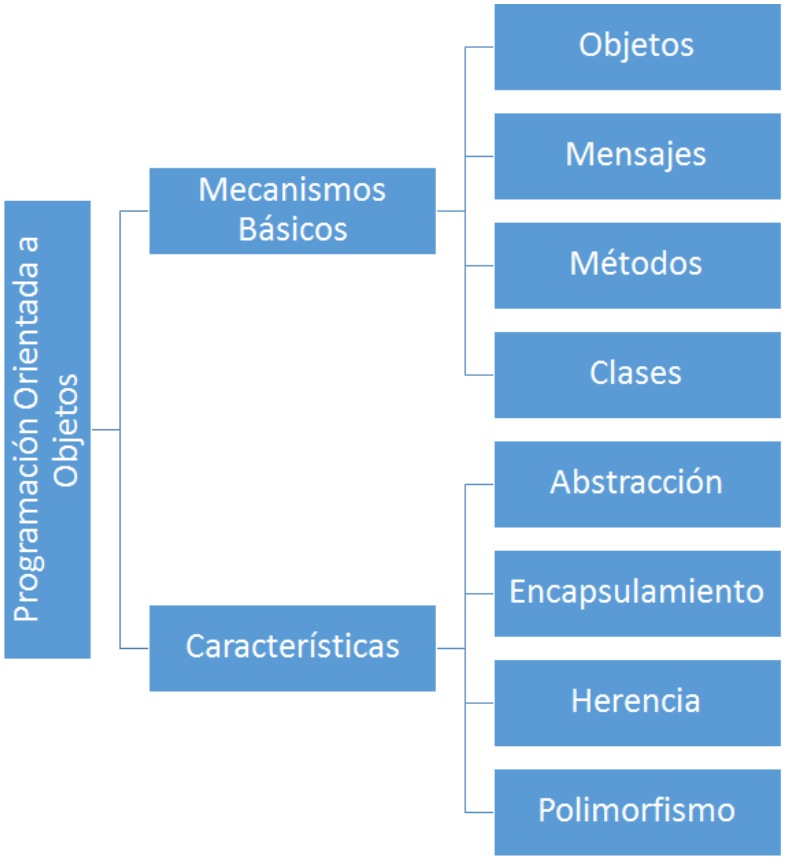
•modalidad **cuadernillos**•

Unidad 1

1.1 Diseña modelos bajo el enfoque de la metodología orientada a objetos.
Actividades según programa de estudios Conalep

Programación Orientada a Objetos

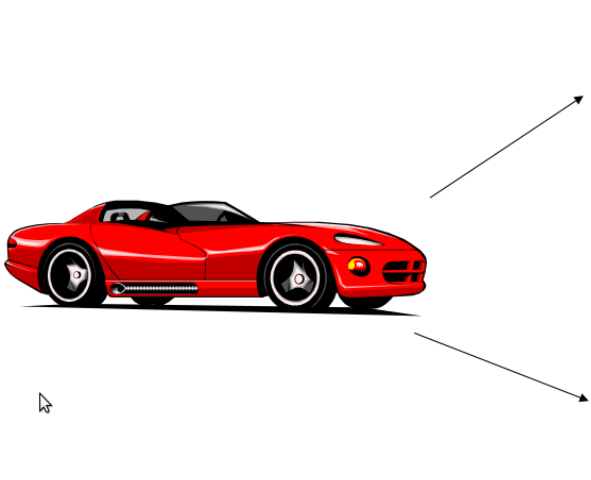
Es un paradigma de programación que define los programas en términos de "clases de objetos", objetos que son entidades que combinan estado (propiedades o datos), comportamiento (procedimientos o métodos) e identidad (propiedad del objeto que lo diferencia del resto).



¿Cuáles son las ventajas de un lenguaje orientado a objetos?

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Facilita la creación de programas visuales.
- Construcción de prototipos
- Agiliza el desarrollo de software
- Facilita el trabajo en equipo
- Facilita el mantenimiento del software

Objeto



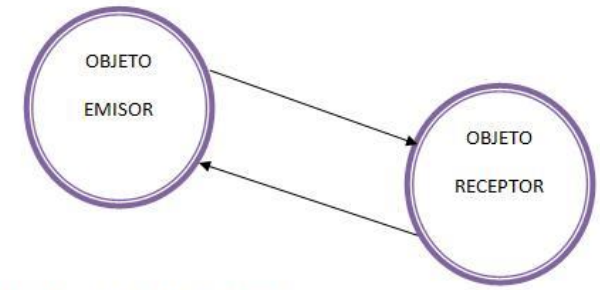
Es un conjunto de variables (o datos) y métodos (o funciones) relacionados entre sí.

Los objetos en programación se usan para modelar objetos o entidades del mundo real (el objeto hijo, madre, o farmacéutica, por ejemplo).

Un objeto es, por tanto, la representación en un programa de un concepto, y contiene toda la información necesaria para abstraerlo: datos que describen sus atributos y operaciones que pueden realizarse sobre los mismos.

Mensaje

Es la petición de un objeto a otro para solicitar la ejecución de alguno de sus métodos o para obtener el valor de un atributo público.
En un mensaje siempre hay un receptor, lo cual no ocurre en una llamada a procedimiento.



Paso de mensajes entre objetos

Un mensaje consta de 3 partes:

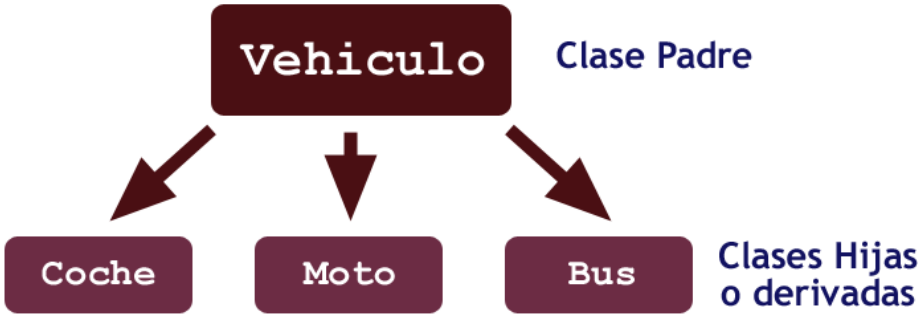
1. Identidad del receptor: Nombre del objeto que contiene el método a ejecutar.
2. Nombre del método a ejecutar: Solo los métodos declarados públicos.
3. Lista de Parámetros que recibe el método (cero o más parámetros)

Herencia

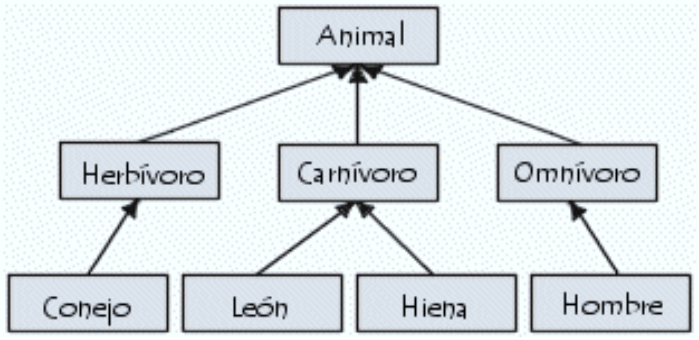
Es una propiedad que permite que los objetos sean creados a partir de otros ya existentes, obteniendo características (métodos y atributos) similares a los ya existentes.

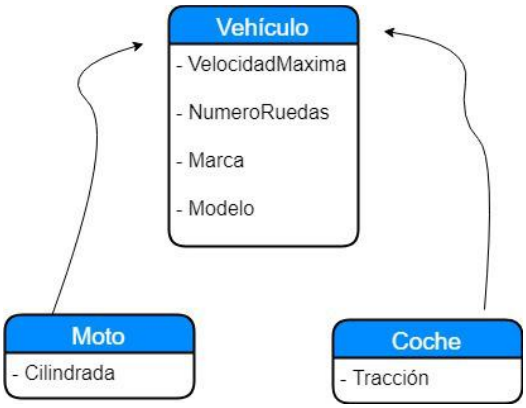
Es la relación entre una clase general y otra clase más específica.

Es un mecanismo que nos permite crear clases derivadas a partir de clase base, nos permite compartir automáticamente métodos y datos entre clases subclases y objetos.



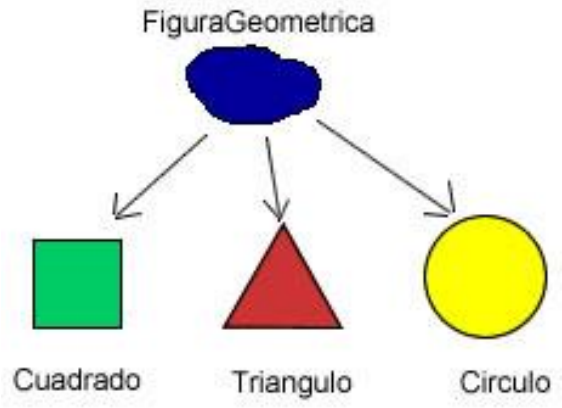
Ejemplos de Herencia



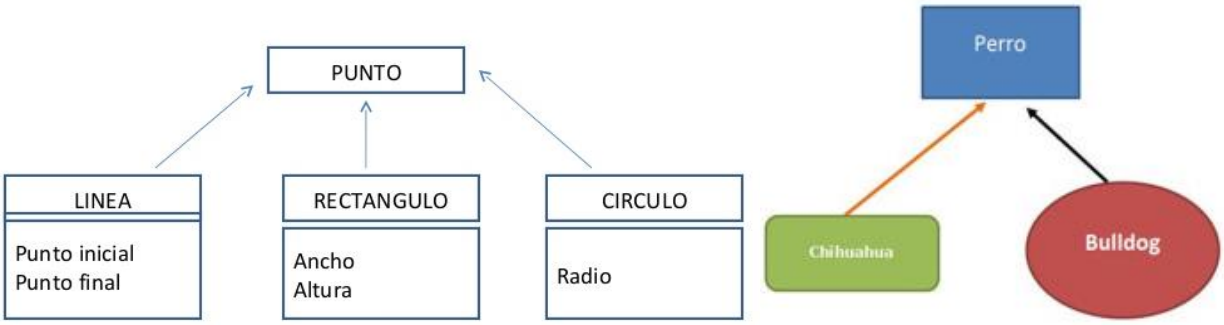


Polimorfismo

Se refiere a la posibilidad de definir clases diferentes que tienen métodos o atributos denominados de forma idéntica, pero que se comportan de manera distinta.
El concepto de polimorfismo se puede aplicar tanto a funciones como a tipos de datos. Así nacen los conceptos de funciones polimórficas y tipos polimórficos. Las primeras son aquellas funciones que pueden evaluarse o ser aplicadas a diferentes tipos de datos de forma indistinta; los tipos polimórficos, por su parte, son aquellos tipos de datos que contienen al menos un elemento cuyo tipo no está especificado.



Ejemplos de Polimorfismo

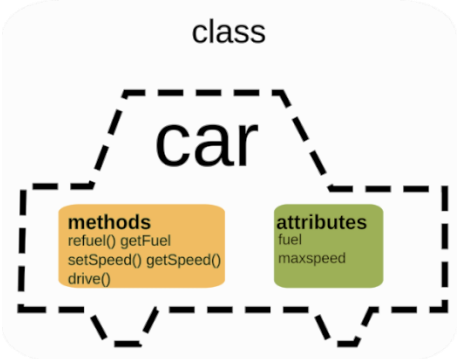


Clase

Es un molde del que luego se pueden crear múltiples objetos, con similares características.
Es una plantilla (molde), que define atributos (variables) y métodos (funciones)
Define los atributos y métodos comunes a los objetos de ese tipo, pero luego, cada objeto tendrá sus propios valores y compartirán las mismas funciones.

La estructura de una clase es:

```
class [nombre de la clase] {
  [atributos o variables de la clase]
  [métodos o funciones de la clase]
  [main]
}
```



Actividad

A. Crea los siguientes objetos en hoja blanca definiendo Atributos y Métodos

- Silla
- Perro
- Lápiz
- Celular
- Platillos
- Taza
- Reloj

B. Crea el diagrama para representar la herencia del objeto que se menciona (Como en el ejemplo de herencia Vehículo).

- Persona
- Animales
- Aves

Modelos para Diseño

Es una técnica de especificación semiformal para el paradigma orientado a objetos. Ya que se trata de una técnica semiformal, una parte intrínseca es la notación gráfica asociada.
El Lenguaje de Modelado Unificado (UML, Unified Modeling Language) se ha desarrollado en un intento de unificar las distintas notaciones existentes.
Se ocupa de comprender y analizar la aplicación y el dominio en el que opera. El punto de partida es la declaración del problema que hay que resolver. Esta declaración, que proporciona una visión conceptual del sistema propuesto, puede ser textual o utilizar una técnica de descripción más formal, como la basada en casos de uso. El modelado orientado a objetos consta de tres pasos: modelado de casos de uso, modelado de clases y modelado dinámico.

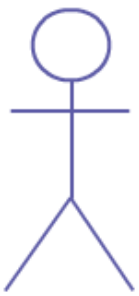
Diagrama Casos de Uso

El diagrama de caso de uso es un tipo de diagrama UML de comportamiento y se usa frecuentemente para analizar varios sistemas. Permiten visualizar los diferentes tipos de roles en un sistema y cómo esos roles interactúan con el sistema. Este tutorial de diagramas de casos de uso cubrirá los siguientes temas y le ayudará a crear mejor los casos de uso.
Se utilizan para reunir los requisitos de uso de un sistema. Dependiendo de sus necesidades, puede utilizar esos datos de diferentes maneras. A continuación, se presentan algunas formas de usarlas.

- Identificarlas funciones y la forma en que los roles interactúan con ellas – El propósito principal de los diagramas de casos de uso.
- Para una visión de alto nivel del sistema – Especialmente útil cuando se presenta a los administradores o a las partes interesadas. Se pueden destacar los papeles que interactúan con el sistema y la funcionalidad proporcionada por el sistema sin profundizar en el funcionamiento interno del sistema.
- Identificar los factores internos y externos – Esto puede parecer simple, pero en grandes proyectos complejos un sistema puede ser identificado como una función externa en otro caso de uso.

Usar los objetos del Diagrama de Caso

Los diagramas de caso de uso consisten en 6 objetos.

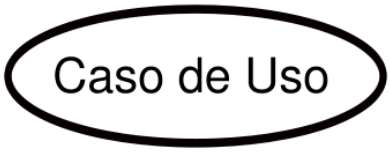


1. **Actor**

El actor en un diagrama de caso de uso de cualquier entidad que desempeñe un papel en un sistema determinado. Puede ser una persona, una organización o un sistema externo y normalmente se dibuja como el esqueleto que se muestra a continuación.

2. **Caso de uso**

Un caso de uso representa una función o una acción dentro del sistema. Está dibujado como un óvalo y nombrado con la función.



3. **Límite de Sistema**

Muestra el límite del sistema que estamos representando, recordemos que en un requerimiento pueden interactuar distintos agentes y distintos casos de uso.

4. **Línea de asociación**

Representa la relación entre un actor y un caso de uso, la relación entre actores y casos de uso puede ser de 1 a 1, de 1 a muchos y de muchos a 1.

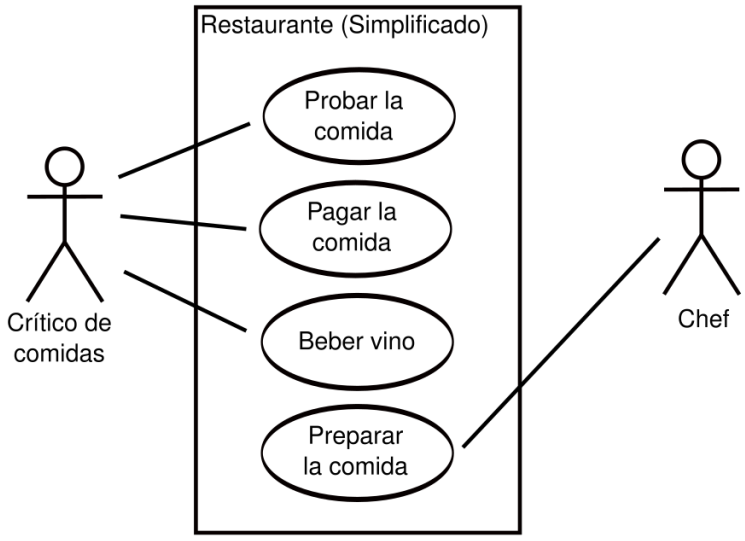
5. **Extensión**

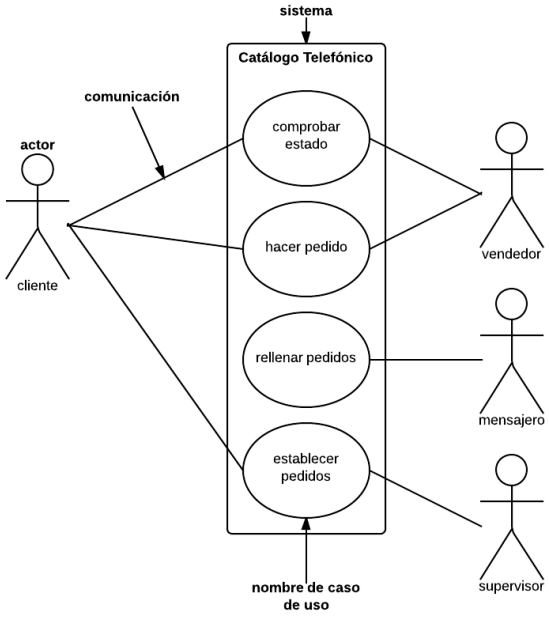
Hace referencia cuando un caso de uso se extiende a otro caso de uso, en el caso del ejemplo que usábamos en caso de uso haría referencia al extends entre verificación y validación. También conocido como extends.

6. **Incluye**

Este proceso hace referencia cuando un caso de uso incluye todos los atributos de otro caso de uso. Por ejemplo, cuando nos suscribimos a una revista deportiva, hay un caso de uso que se llamará inscripción revista y este caso de uso incluirá un caso de uso que se llaman revistas deportivas.

Ejemplo de caso de Uso





Actividad

Instrucciones: Da solución a los siguientes ejercicios de Diagrama de Casos de Uso

Problema No. 1. Modelado de diagrama de casos de uso de una tienda de electrodomésticos.

Planteamiento:

Un negocio de venta de electrodomésticos decidió implementar y otorgar una línea de crédito a sus clientes para la compra de productos.
Los créditos son solicitados por los clientes al vendedor al momento de realizar la compra y deben ser autorizados por un representante de la gerencia de créditos, y pagados por el cliente a través del débito automático en tarjetas de crédito. Si el crédito se acepta, se entrega el producto al cliente en forma inmediata.
Cada mes se cobrará de manera automática el pago de las cuotas de la tarjeta del cliente.
Se quiere modelar el proceso de solicitud, otorgamiento, y pago del crédito.

a. Represente todo el proceso completo como si fuera un único caso de uso, mencionando sus pasos más importantes, sin entrar en detalles sobre alternativas.
b. Identifique los distintos actores que intervienen en este proceso.
c. Teniendo en cuenta su resolución del primer punto, identifique casos de uso de este proceso, que pueden ser las distintas partes del caso completo presentado en el punto a.
d. A partir de los casos ya identificados, pensando en casos anteriores, siguientes, contrarios o que sean variaciones de los mismos, identifique nuevos casos de uso o alternativas entre los casos.
e. Identifique casos que puedan ser extraídos de los anteriores y ser "usados" por otros casos.

Problema No. 2. Modelado de diagrama de casos de uso de una cadena de videoclubes.

Planteamiento:

La famosa cadena de videoclubes "Los Bloques de Búster" nos ha contratado con el fin de desarrollar un sistema para sistematizar sus locales.
Hasta el día de hoy se han mantenido una serie de reuniones con el cliente con el fin de determinar los requerimientos del sistema. De tales reuniones, se ha determinado lo siguiente:
El sistema deberá permitir que los clientes consulten el catálogo de películas. A partir del mismo, una vez seleccionada una película, se deberá poder acceder a la información de la misma como ser su clasificación, su género y un breve resumen de la misma. Asimismo, opcionalmente, se deberá poder consultar la disponibilidad del video.
Los empleados del videoclub deberán poder, a través del sistema, registrar las rentas y devoluciones por parte de los clientes, y consultar, dado un cliente, los videos que éste posea en renta. Si registrando una rentar, resulta que el cliente no se encuentra registrado, el sistema deberá permitir que se efectúe su alta.
Nuestro cliente también pidió que el sistema, todas las mañanas genere de forma automática un informe que muestre todos los clientes que se encuentran atrasados con sus devoluciones. Cuando se le preguntó a qué se refería con "todas las mañanas" aclaró: "Que todos los días a las 9:00 a.m. imprima o muestre por pantalla el listado de los clientes atrasados."

Problema No. 3. Modelado de diagrama de casos de uso de una máquina expendedora y de venta de bebidas.

Planteamiento:

Se ha decidido fabricar una máquina para la expedición y venta de bebidas en forma automática.

El cliente selecciona algunos de los productos ofrecidos, uno o más, por medio de la pulsación de uno o más botones. Los artículos pueden ser de distintos tipos: latas de refresco, jugos o botellas.

Solamente se puede solicitar un tipo de producto a la vez. La máquina reconoce el pedido del cliente. Si no hay en existencia le indica al cliente por medio de un mensaje.

La máquina acepta las monedas del cliente, reconociendo de distintos tipos. Si las monedas no cubren el total del importe las devuelve y le avisa al cliente por medio de un mensaje. En caso contrario, libera las bebidas solicitadas, actualiza el stock de productos e imprime un ticket.

El encargado de la reposición, repone los artículos de acuerdo a lo indicado en la pantalla (tendrá una pantalla propia, a la que accederá mediante su contraseña). Al hacerlo, debe indicarle al sistema el producto y la cantidad que se ha repuesto. Inmediatamente el sistema deberá actualizar el stock, emitir un resumen de faltante en dos copias, como constancia de reposición y factura para el poseedor de la máquina

Diagrama de Clases

Sirve para visualizar las relaciones entre las clases que involucran el sistema, las cuales pueden ser asociativas, de herencia, de uso.

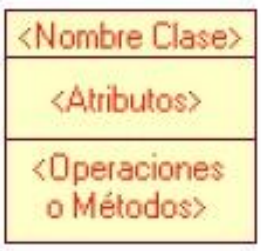
Un diagrama de clases está compuesto por los siguientes elementos:

- **Clase:** atributos, métodos y visibilidad.
- **Relaciones:** Herencia, Composición, Agregación, Asociación y Uso.

Clase

Es la unidad básica que encapsula toda la información de un Objeto (un objeto es una instancia de una clase). A través de ella podemos modelar el entorno en estudio (una Casa, un Auto, una Cuenta Corriente, etc.).

En UML, una clase es representada por un rectángulo que posee tres divisiones:



En donde:

Superior: Contiene el nombre de la Clase

Intermedio: Contiene los atributos (o variables de instancia) que caracterizan a la Clase (pueden ser private, protected o public).

Inferior: Contiene los métodos u operaciones, los cuales son la forma como interactúa el objeto con su entorno (dependiendo de la visibilidad: private, protected o public).

Relaciones

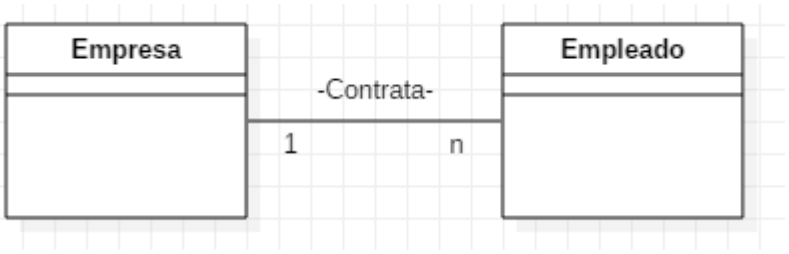
Una relación identifica una dependencia. Esta dependencia puede ser entre dos o más clases (más común) o una clase hacía sí misma (menos común, pero existen), este último tipo de dependencia se denomina

dependencia reflexiva. Las relaciones se representan con una línea que une las clases, esta línea variará dependiendo del tipo de relación

Las relaciones en el diagrama de clases tienen varias propiedades, que dependiendo la profundidad que se quiera dar al diagrama se representarán o no. Estas propiedades son las siguientes:

- **Multiplicidad.** Es decir, el número de elementos de una clase que participan en una relación. Se puede indicar un número, un rango… Se utiliza n o * para identificar un número cualquiera.

- **Nombre de la asociación.** En ocasiones se escriba una indicación de la asociación que ayuda a entender la relación que tienen dos clases. Suelen utilizarse verbos como, por ejemplo: "Una empresa contrata a n empleados"

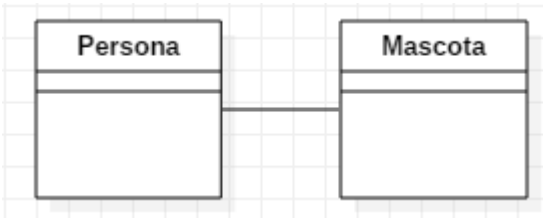


Tipos de relaciones

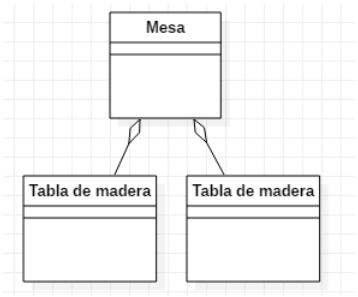
Un diagrama de clases incluye los siguientes tipos de relaciones:

- **Asociación**

Este tipo de relación es el más común y se utiliza para representar dependencia semántica. Se representa con una simple línea continua que une las clases que están incluidas en la asociación. Un ejemplo de asociación podría ser: "Una mascota pertenece a una persona".



- **Agregación.**

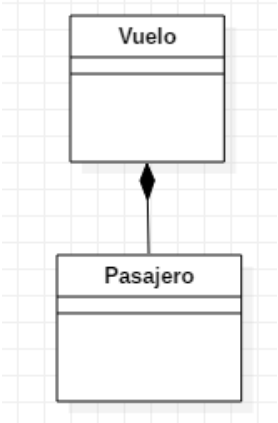


Es una representación jerárquica que indica a un objeto y las partes que componen ese objeto. Es decir, representa relaciones en las que un objeto es parte de otro, pero aun así debe tener existencia en sí mismo.

Se representa con una línea que tiene un rombo en la parte de la clase que es una agregación de la otra clase (es decir, en la clase que contiene las otras).

- **Composición.**

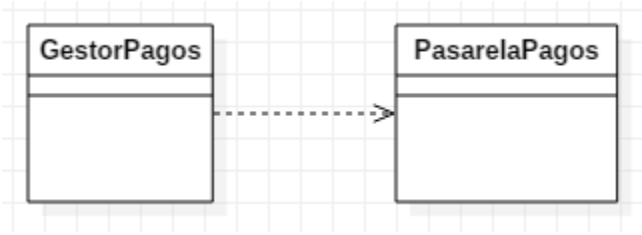
La composición es similar a la agregación, representa una relación jerárquica entre un objeto y las partes que lo componen, pero de una forma más fuerte. En este caso, los elementos que forman parte no tienen sentido de existencia cuando el primero no existe. Es decir, cuando el elemento que contiene los otros desaparece, deben desaparecer todos ya que no tienen sentido por sí mismos, sino que dependen del elemento que componen. Los componentes no se comparten entre varios elementos, esta es otra de las diferencias con la agregación.



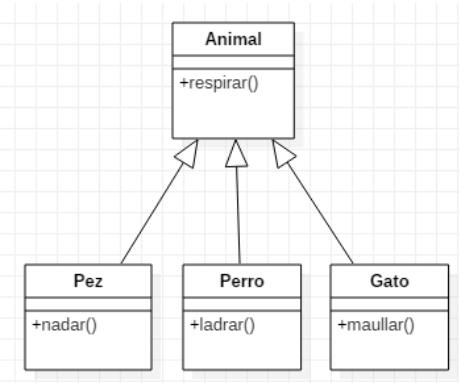
Se representa con una línea continua con un rombo relleno en la clase que es compuesta.
Un ejemplo de esta relación sería: "Un vuelo de una compañía aérea está compuesto por pasajeros, que es lo mismo que decir que un pasajero está asignado a un vuelo"

- **Dependencia.**

Se utiliza este tipo de relación para representar que una clase requiere de otra para ofrecer sus funcionalidades. Es muy sencilla y se representa con una flecha discontinua que va desde la clase que necesita la utilidad de la otra flecha hasta esta misma.
Un ejemplo de esta relación podría ser la siguiente:



- **Herencia.**



Este tipo de relaciones permiten que una clase (clase hija o subclase) reciba los atributos y métodos de otra clase (clase padre o superclase). Estos atributos y métodos recibidos se suman a los que la clase tiene por sí misma. Se utiliza en relaciones "es un".
Un ejemplo de esta relación podría ser la siguiente: Un pez, un perro y un gato son animales.

Ejemplos de diagrama de clases

Diagrama de clases clínica veterinaria

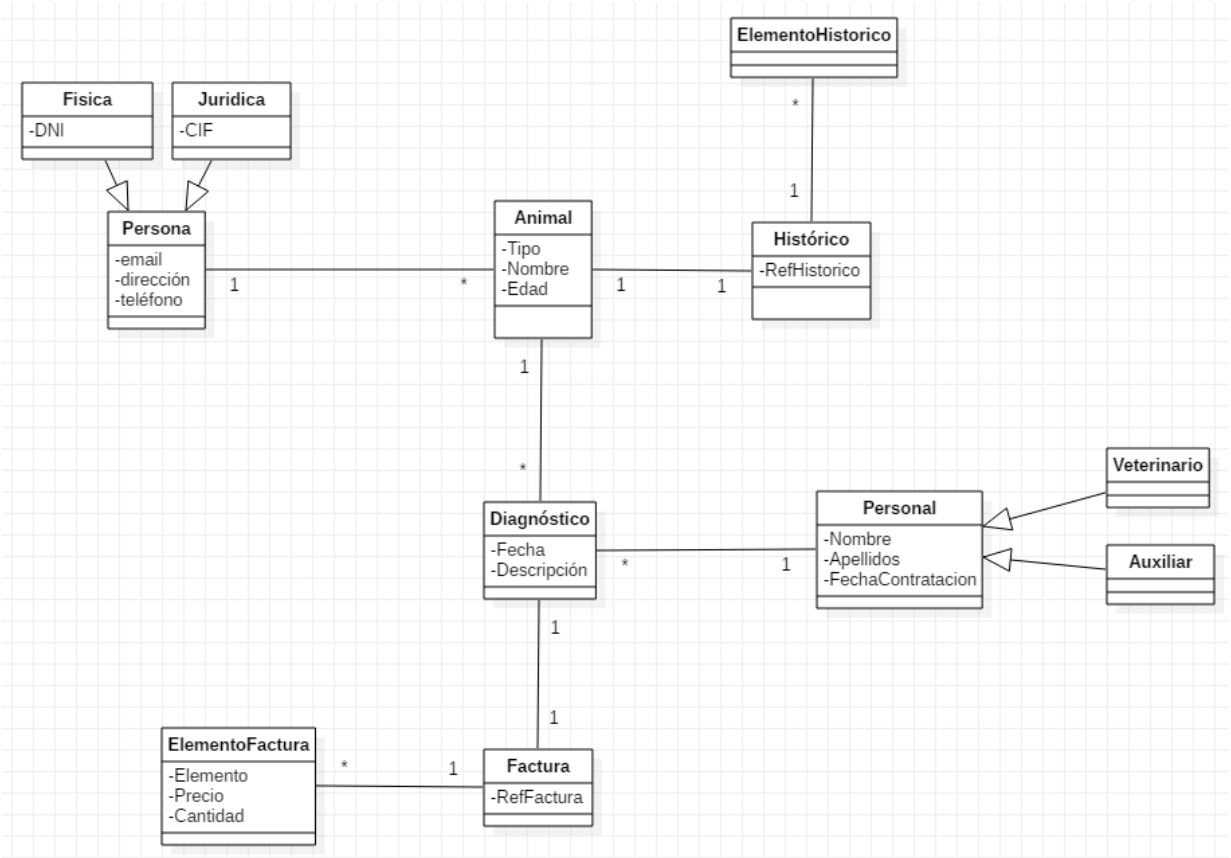


Diagrama de clases de una tienda

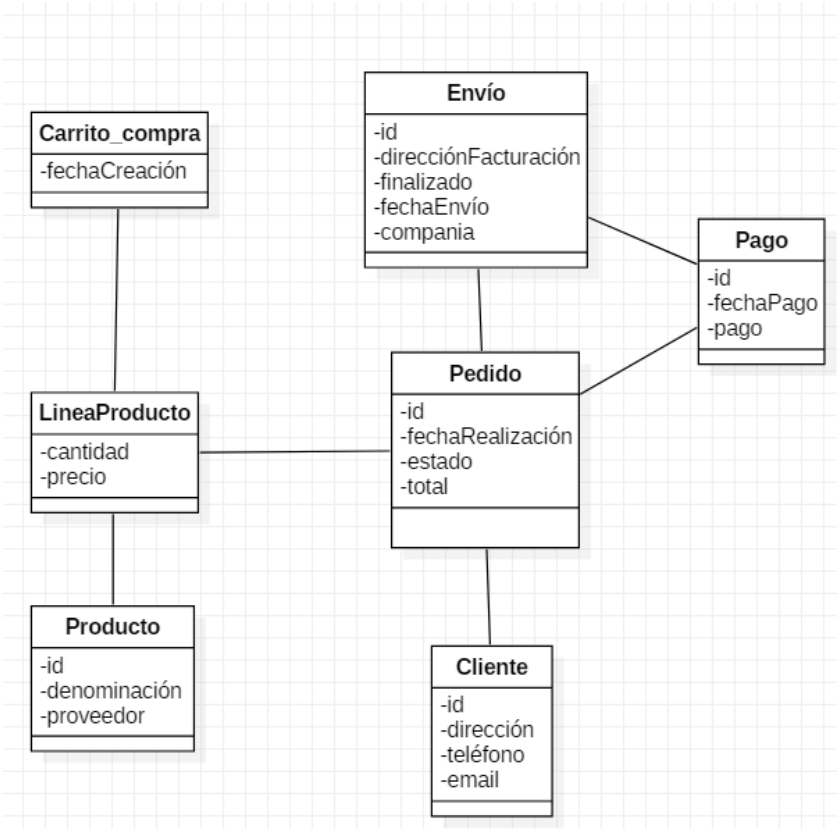
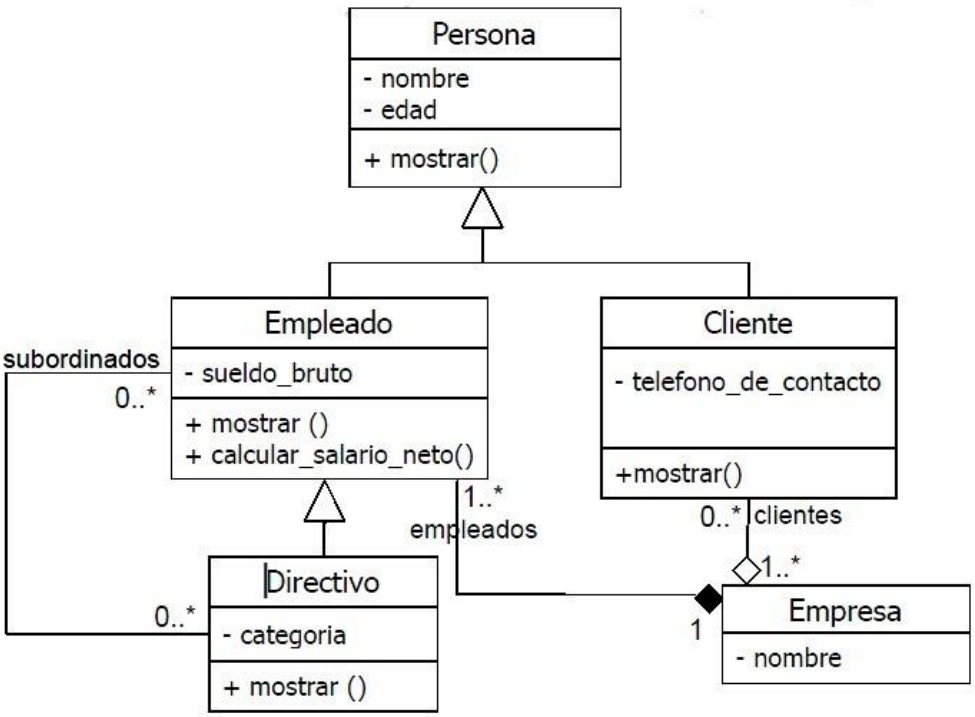


Diagrama de clases de una empresa



Actividad

Instrucciones: Da solución a los siguientes ejercicios de Diagrama

Problema No. 4 Modelado de diagrama de clases del evento del comité olímpico Internacional.

Planteamiento:

Usted ha sido contratado por el COI (Comité Olímpico Internacional) para analizar, diseñar e implementar una solución que permita al comité tener conocimiento de todos los detalles implicados a este gran evento. Luego de reiteradas reuniones con el COI se decidió efectuar un desarrollo piloto sobre las competencias de fútbol de las Olimpíadas, a fin de determinar la efectividad del futuro sistema.
Las competencias de fútbol contarán con una serie de estadios para el desarrollo de cada uno de los partidos. Cada partido contará con la participación de dos equipos los cuales serán representación de un país invitado y un grupo designado de árbitros; cada uno de estos partidos debe proporcionarnos información acerca de: los goles marcados y las tarjetas sacadas, además de saber cuál fue la asistencia de público a cada partido y la fase de evento a la cual pertenecía. Cada equipo está conformado por 20 jugadores y un cuerpo técnico. De cada jugador queremos saber información como su nombre, fecha y lugar de nacimiento, posición que juega, etc.
Del cuerpo técnico es necesario conocer su nombre, fecha y lugar de nacimiento, cargo, etc.
De los árbitros se necesita saber su nombre, fecha y lugar de nacimiento, federación a la que pertenece y su cargo.
De los goles y las tarjetas queremos conocer el partido, el minuto y la persona que está relacionado con el gol (quién lo metió). Los estadios de las Olimpiadas son estadios que en su mayoría se han reformado o construido nuevos para darle un impulso al deporte en China. Es por ello que el comité organizador desea saber las características básicas de cada estadio como, por ejemplo, capacidad, ciudad donde están localizados, si posee techo o no, etc.

Problema No. 5 Modelado de diagrama de clases de un estacionamiento.

Planteamiento:

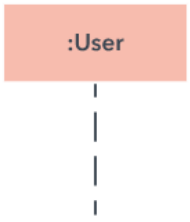
Se desea automatizar un estacionamiento con capacidad para 400 automóviles, de acuerdo a los siguientes requisitos:
Los usuarios del estacionamiento dispondrán de una tarjeta magnética donde figura registrado su código de identificación.
A su llegada al estacionamiento, el usuario introducirá la tarjeta en el lector correspondiente, lo que hace que se eleve la barrera situada en la entrada. Esta barrera permanece levantada un cierto tiempo, descendiendo luego automáticamente.
Para salir del aparcamiento se procede de igual forma con la barrera situada a la salida.
Tanto las entradas como las salidas deben quedar registradas con objeto de realizar periódicamente una facturación a los usuarios, según el tiempo de estacionamiento consumido. Estas facturas se emitirán a petición del operador.
El sistema debe tener en cuenta la ocupación del estacionamiento, controlando un semáforo situado a la entrada. Si hay lugares disponibles libres, el semáforo debe estar verde, pasando a rojo si el estacionamiento se llena. Además, cuando el aparcamiento esté lleno no debe permitirse la entrada a nuevos vehículos.

Diagrama de Secuencia

Es un tipo de diagrama de interacción porque describe cómo —y en qué orden— un grupo de objetos funcionan en conjunto. Tanto los desarrolladores de software como los profesionales de negocios usan estos diagramas para comprender los requisitos de un sistema nuevo o documentar un proceso existente. A los diagramas de secuencia en ocasiones se los conoce como diagramas de eventos o escenarios de eventos.Observa que hay dos tipos de diagramas de secuencia: los diagramas UML y los diagramas que se basan en código. Los últimos se obtienen de un código de programación y no serán cubiertos en esta guía. El software de diagramas UML de Lucidchart está equipado con todas las figuras y funciones que necesitarás para modelar ambos.

Componentes y símbolos básicos

Para comprender qué es un diagrama de secuencia, debes estar familiarizado con sus símbolos y componentes. Los diagramas de secuencia están formados por los siguientes elementos e íconos:

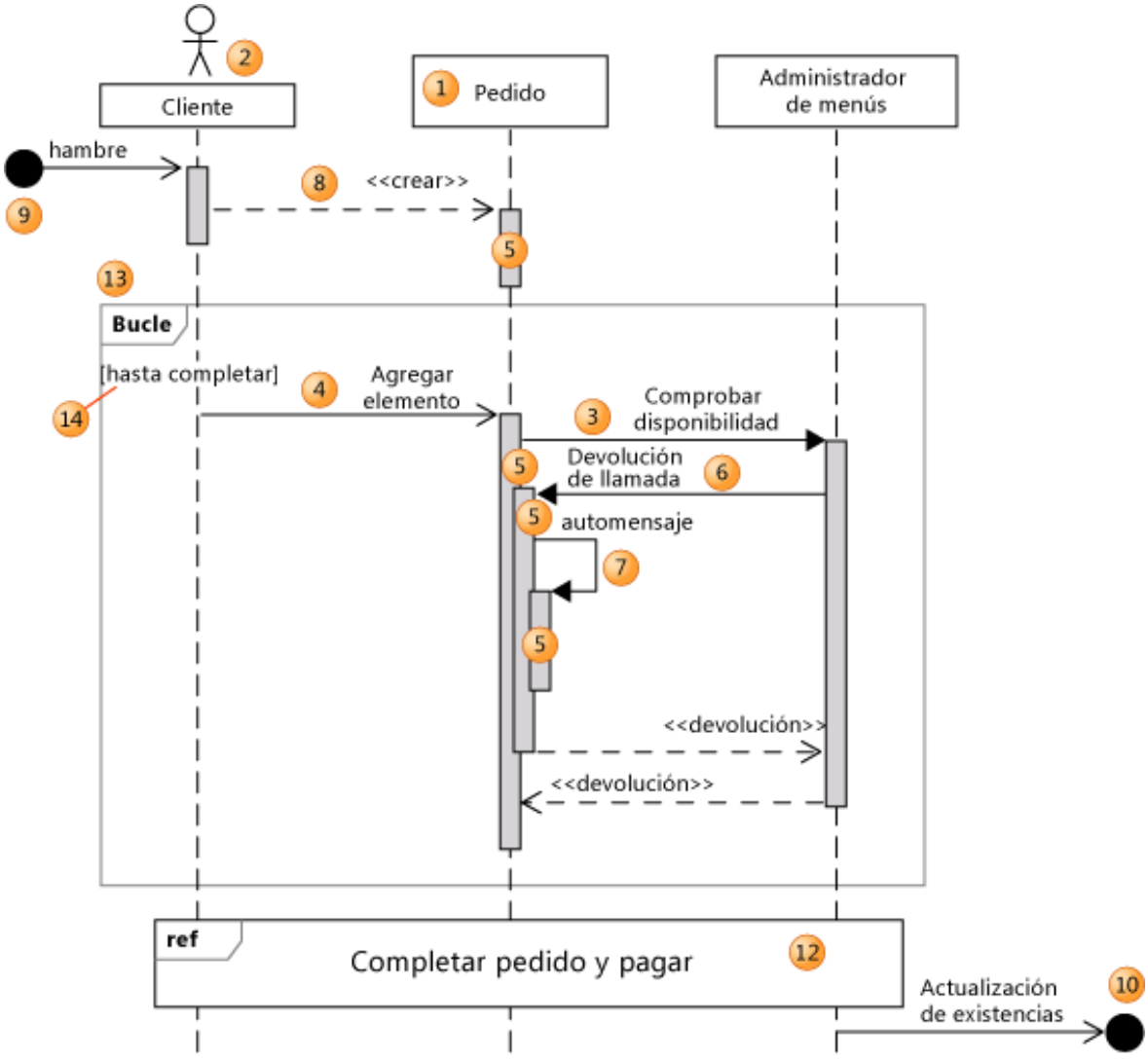
Símbolo	Nombre	Descripción
	Símbolo de objeto	Representa una clase u objeto en UML. El símbolo objeto demuestra cómo se comportará un objeto en el contexto del sistema. Los atributos de las clases no deben aparecer en esta figura.
	Casilla de activación	Representa el tiempo necesario para que un objeto finalice una tarea. Cuanto más tiempo lleve la tarea, más larga será la casilla de activación.
	Símbolo de actor	Muestra entidades que interactúan con el sistema o que son externas al sistema.
	Símbolo de paquete	Se usa en notación UML 2.0 para contener los elementos interactivos del diagrama. También conocida como "marco", esta figura rectangular tiene un pequeño rectángulo interior para etiquetar el diagrama.
	Símbolo de línea de vida	Representa el paso del tiempo a medida que se extiende hacia abajo. Esta línea vertical discontinua representa eventos secuenciales que le ocurren a un objeto durante el proceso graficado. Las líneas de vida pueden comenzar con una figura rectangular etiquetada o un símbolo de actor.

Símbolos comunes de mensajes

Usa los siguientes símbolos de mensaje y flechas para indicar cómo se transmite la información entre objetos. Estos símbolos pueden reflejar el inicio y la ejecución de una operación o el envío y la recepción de una señal.

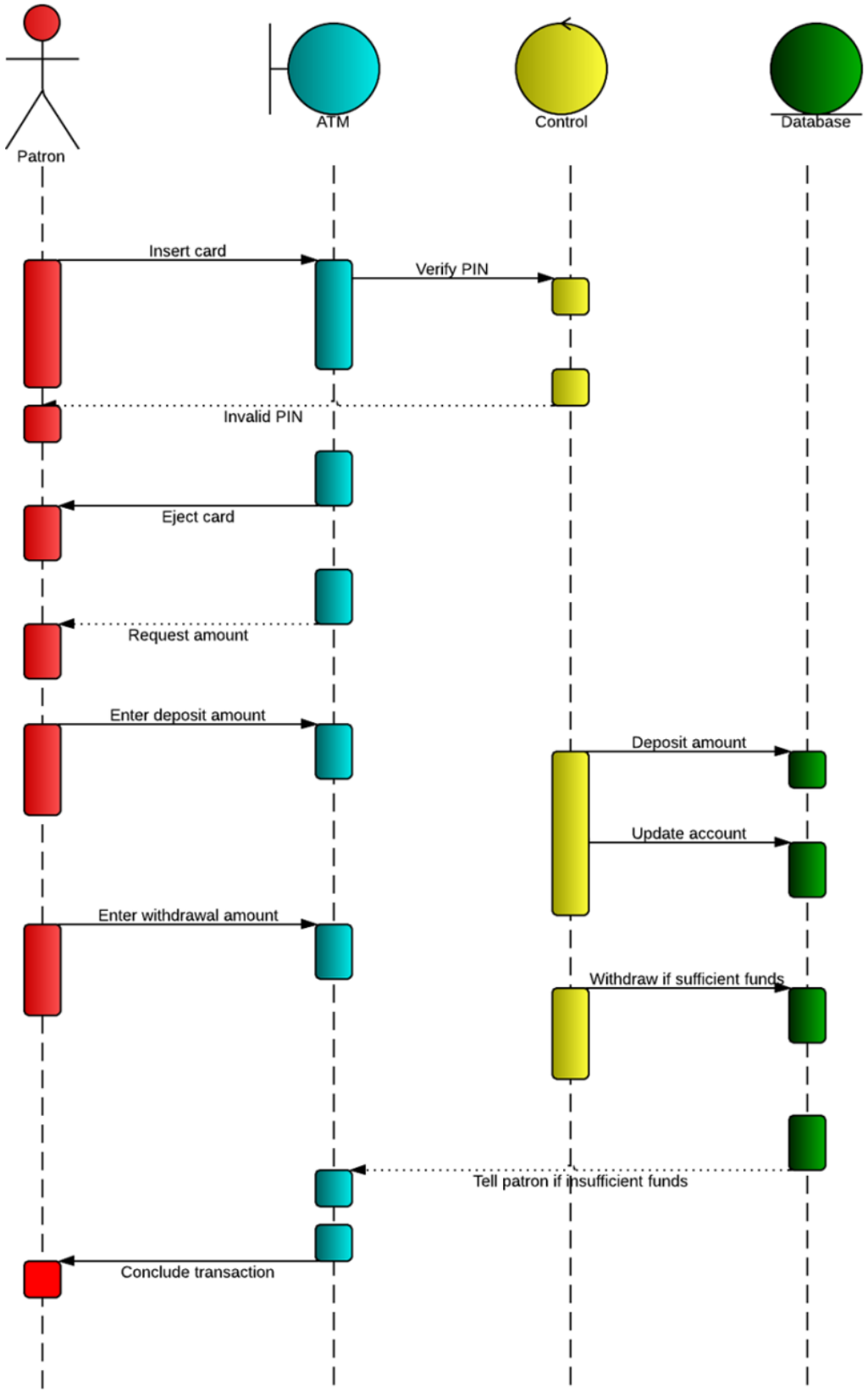
Símbolo	Nombre	Descripción
	Símbolo de mensaje sincrónico	Representados por una línea continua y una punta de flecha sólida. Este símbolo se utiliza cuando un remitente debe esperar una respuesta a un mensaje antes de proseguir. El diagrama debe mostrar el mensaje y la respuesta.
	Símbolo de mensaje asincrónico	Representados por una línea continua y una punta de flecha simple. Los mensajes asincrónicos no necesitan una respuesta para que el remitente siga adelante. Solo la llamada se debe incluir en el diagrama.
	Símbolo de mensaje de respuesta asincrónico	Representados por una línea discontinua y una punta de flecha simple.
	Símbolo de mensaje de respuesta	Están representados con una línea discontinua y una punta de flecha simple. Estos mensajes son las respuestas a las llamadas.

Ejemplo de Diagrama de Secuencia Realizar un Pedido



Ejemplo de Diagrama de secuencia para sistemas de cajero automático ATM

Un cajero ATM permite a los clientes acceder a sus cuentas bancarias a través de un proceso completamente automatizado. Puedes examinar los pasos de este proceso de una forma manejable dibujando o visualizando un diagrama de secuencia. El siguiente ejemplo describe el orden secuencial de las interacciones en el sistema ATM.



Actividad

Instrucciones: Da solución a los siguientes ejercicios de Diagrama

Problema No. 6 Modelado de diagrama de secuencia de un centro de instalaciones deportivas.

Planteamiento:

Un centro de instalaciones deportivas quiere hacer una aplicación de reservas. En el centro existen instalaciones deportivas (piscinas, frontones, gimnasios y pistas de tesis). El centro en cuestión tiene socios, de los cuales se almacenan su nombre, dirección, ciudad, estado, teléfono y cuota.
Además, existen una serie de artículos que se pueden reservar si el socio lo requiere (balones, redes y raquetas).
Cada instalación es reservada por un socio en una fecha dada desde una hora de inicio hasta una hora de fin.
Cada reserva puede tener asociada uno o varios artículos deportivos que se alquilan aparte. Por ejemplo, si yo quiero hacer una reserva para jugar al tenis, tengo que reservar una instalación deportiva y si lo necesito, las raquetas.

Problema No. 7 Modelado de diagrama de secuencia de un controlador de pistas scalestrix.

Planteamiento:

Se desea realizar un controlador para una pista de autos scalextric.
Dicho scalextric estará formado por una pista por la que circularán dos vehículos, de forma que uno será controlado por computador, y el otro por una persona.
Los vehículos tendrán asociado un identificador, una velocidad, un tiempo, un contador de vueltas, y un sensor que informe de la proximidad del vehículo contrincante.
A escasos centímetros del comienzo de una curva existirán unos sensores que informen acerca de la curvatura de la misma al vehículo.
De la misma forma, a pocos centímetros de una pendiente, también estarán colocados sensores que informen al vehículo de la inclinación de la misma.
Estos dos tipos de sensores le servirán para calcular la velocidad a tomar en el siguiente instante.
En la pista también habrá un sensor en la línea de meta para que el coche pueda llevar la cuenta de las vueltas.
El final de la carrera se determinará por el cumplimiento de una serie de vueltas configuradas a principio de la carrera, o por la salida de la pista de uno de los dos vehículos.
En ambos casos aparecerá en una pantalla el coche ganador. Si ninguno de los dos se ha salido de la pista, se incluirá el identificador de este coche junto con la duración en una tabla de récords.

Software para el desarrollo y modelado de aplicaciones orientada a objetos.

JAVA

Es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Python

Es un lenguaje de programación poderoso y fácil de aprender. Cuenta con estructuras de datos eficientes y de alto nivel y un enfoque simple pero efectivo a la programación orientada a objetos. La elegante sintaxis de Python y su tipado dinámico, junto con su naturaleza interpretada, hacen de éste un lenguaje ideal para scripting y desarrollo rápido de aplicaciones en diversas áreas y sobre la mayoría de las plataformas.

Ruby

Es un "lenguaje de guiones (scripts) para una programación orientada a objetos rápida y sencilla". Tiene la posibilidad de realizar directamente llamadas al sistema operativo, potentes operaciones sobre cadenas de caracteres y expresiones regulares, retroalimentación inmediata durante el proceso de desarrollo, son innecesarias las declaraciones de variables, las variables no tienen tipo, la sintaxis es simple y consistente, la gestión de la memoria es automática.

Smalltalk

Es un lenguaje reflexivo de programación, orientado a objetos y con tipado dinámico. Por sus características, Smalltalk puede ser considerado también como un entorno de objetos, donde incluso el propio sistema es un objeto. Metafóricamente, se puede considerar que un Smalltalk es un mundo virtual donde viven objetos que se comunican entre sí, mediante el envío de mensajes.

Actividad

1. Realizar una investigación documental, para analizar y describir al menos 3 lenguajes de programación que proveen soporte para el desarrollo de aplicaciones orientadas a objetos en el que describa:
    - Plataforma
    - Editor de texto expleado.
    - Compilador.
    - Enlazador.
    - Entornos de Desarrollo Integrado.

2. Tabla descriptiva de características de lenguajes de programación orientada a objeto.

Unidad 1

1.2 Modela y codifica programas de cómputo haciendo uso del paradigma orientado a objetos, a través del uso de sentencias de control, objetos y clases

Actividades según programa de estudios Conalep

Manejo de Sentencias de Control

Interacción del usuario con el programa.

Salida: Escribir en la pantalla. **System.out.println("Hola");**
Entrada: Leer del teclado.
import java.util.Scanner; **//Biblioteca**
Scanner teclado = new Scanner(System.in); **//Declaración de variable para ingresar datos**
String nombre = teclado.nextLine(); **//Declaración variable tipo cadena**
int horas = teclado.nextInt(); **//Declaración variable tipo entero**
double precio = teclado.nextDouble(); **//Declaración variable tipo doble**

¿Qué es el flujo de un programa?

- Es el orden en que se ejecutan las instrucciones de un programa
- El orden normal es instrucción por instrucción, es decir en secuencia.
- El bloque permite esta ejecución secuencial

- Un bloque contiene las instrucciones entre las llaves:

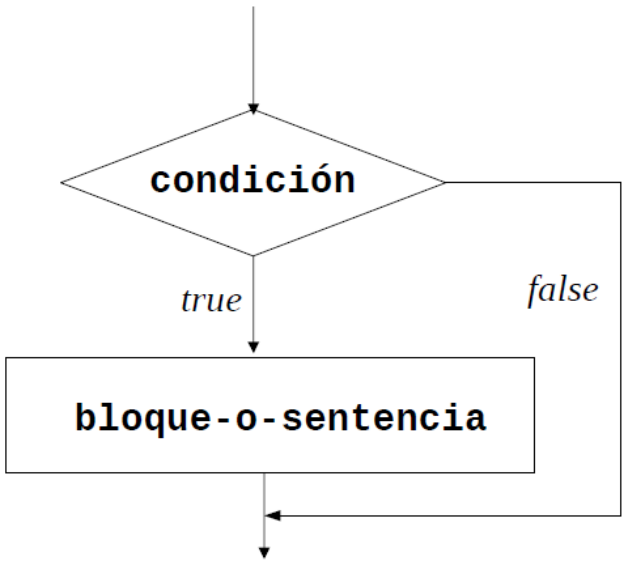
```
{
sentencia 1;
sentencia 2;
sentencia 3;
...
}
```

En Java, se puede modificar el flujo secuencial mediante las estructuras de control:

Estructuras condicionales o selección: un bloque sólo se ejecuta bajo ciertas condiciones

if

Diagrama de flujo



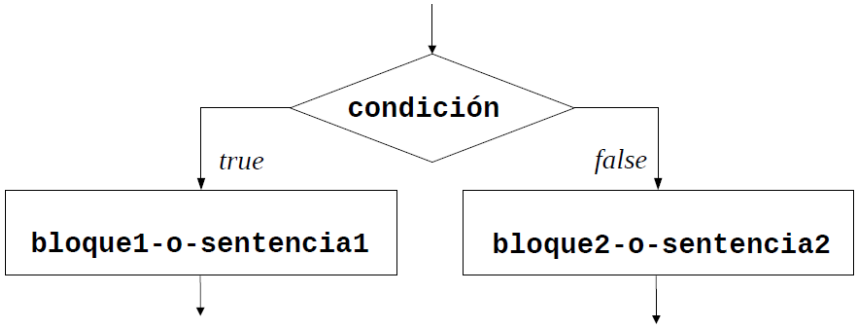
Sintaxis:

```
if (<condición>){
   <bloque-o-sentencia>
      . . . . . .
}
```

Ejemplo de código:

```
import java.util.Scanner;

public class EsDivisible {
//Determinar si un número es divisible por otro
    public static void main(String[] args) {
        int num1, num2;
        Scanner lectura=new Scanner(System.in);
        //lectura del primer número
        System.out.println("Teclea el primer número");
        num1=lectura.nextInt();
        //lectura del segundo número
        System.out.println("Teclea el segundo número");
        num2=lectura.nextInt();
        if (num1%num2==0)
            System.out.println(num1+" es divisible por "+num2);
    }

}
```

if – else



Sintaxis:

```
if (<condición>){
    <bloque1-o-sentencia1>
    }
else{
    <bloque2-o-sentencia2>
    }
```

Ejemplo de código

```java
import java.util.Scanner;

public class EsByte {

    public static void main(String[] args) {
        int num;
        Scanner reader =new Scanner(System.in);
        System.out.println("Teclea número");
        num=reader.nextInt();
        if (num>=-128&&num<=127)
            System.out.println("el número "+num+" es byte");
        else
            System.out.println("error");


    }

}
```

if- else – if

Por ejemplo

```
if (x > 0) {
        if (y > 0){
                System.out.println(“Ambos son mayores que 0");
        }
        else {
                System.out.println(“Alguno no es mayor que 0");
        }
}
```

Switch

- A menudo la condición de un if-else-if depende de una sola variable de tipo primitivo
- Tipos primitivos = palabras reservadas; por ejemplo byte, int, double pero no String
- En este caso se puede utilizar otra instrucción llamada switch
- Es más compacto que un if-else-if

Sintaxis:

```
switch (<selector>) {
        case <etiqueta_1>:
                <sentencias1>;
        break;
        case <etiqueta_2>:
                <sentencias2>;
        break;
        …
        case <etiqueta_n>:
                <sentenciasn>;
        break;
        default:
                <sentenciasD>; // opcional
                break;
}
```

NOTA:

- La función de **break** es cambiar el control de flujo
- ¿Qué pasa si excluimos break? ¡El programa pasa a la siguiente instrucción dentro del switch!

Ejemplo de código utilizando switch

Código que imprime los números Romanos

```
import java.util.Scanner;
public class NumerosRomanos {
        public static void main(String[] args) {
                int num;
                Scanner reader =new Scanner(System.in);
                System.out.println("Teclea número");
                num=reader.nextInt();
                switch (num){
                        case 1:
                                System.out.println("I");
                                break;
                        case 2:
                                System.out.println("II");
                                break;
                        case 3:
                                System.out.println("III");
                                break;
                        case 4:
                                System.out.println("IV");
                                break;
                        case 5:
                                System.out.println("V");
                                break;
                        case 6:
                                System.out.println("VI");
                                break;
                        case 7:
                                System.out.println("VII");
                                break;
                        case 8:
                                System.out.println("VIII")
                                break;
                        case 9:
                                System.out.println("IX");
                                break;
                        case 10:
                                System.out.println("X");
                                break;
                }
        }
}
```

Estructuras de repetición: un mismo bloque se ejecuta repetidamente, a veces hay que ejecutar la misma operación más de una vez.

Ejemplos:

– Escribir ¡Bienvenidos! en la pantalla 10 o 100 o 1000 veces

– Escribir todas las letras del abecedario en la pantalla

– Sumar los gastos anuales de una empresa

En cada caso, se pueden escribir las mismas instrucciones secuencialmente (esto es muy ineficaz)

En cambio, podemos usar bucles (estructuras de repetición)

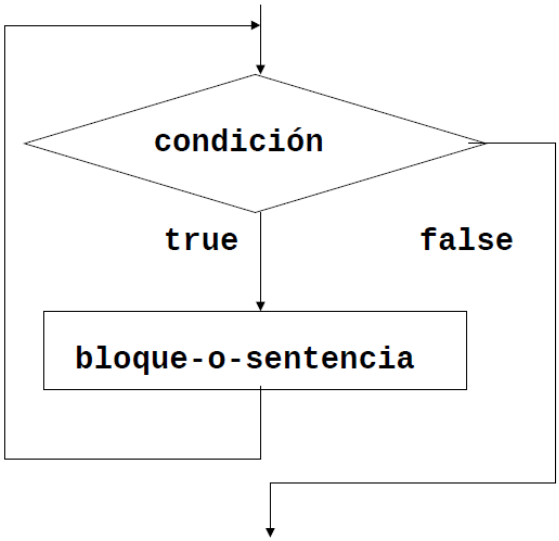
While

La condición de un **while** sirve para determinar cuándo debe terminar el bucle.

¡Si la condición siempre se cumple (true), el bucle while nunca termina! (bucle infinito) En consecuencia, algo en la condición tiene que cambiar.

Cada bucle necesita una variable que aparezca en la condición y cuyo valor cambie en cada pasada. Hay que inicializar el valor de la variable de bucle y actualizar su valor dentro del bucle.

Diagrama de flujo



Sintaxis java:

```
while (<condición>){
    <bloque-o-sentencia>
}
```

Por ejemplo:

```
int contador = 1; // inicialización
while (contador < 6) { // condición
        System.out.println(contador);
        contador++; // actualización
}
```

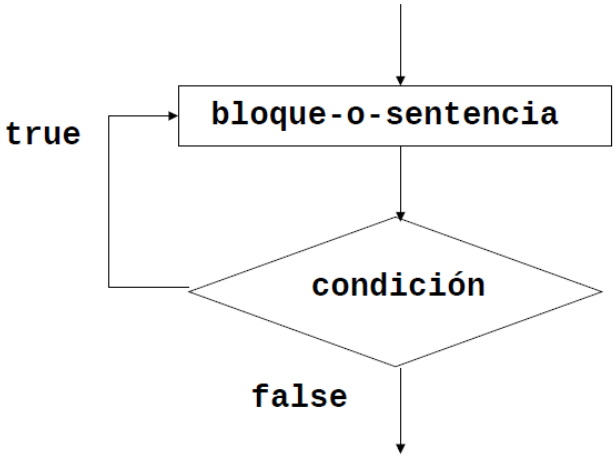
Ejemplo 2:

```
public class Bienvenidos {
        public static void main(String[] args) {
                int contador = 0;
                while (contador < 10) {
                        System.out.println("Bienvenid
                        os");
                        contador++;
                }
        }
}
```

Do while

La diferencia entre un bucle while y un bucle do-while es que el bloque del do-while siempre ***se ejecuta por lo menos una vez***., la condición se prueba al final del bucle y siempre se puede convertir en un while, pero hay casos en que el do-while es más compacto.

Diagrama de Flujo



Sintaxis java:

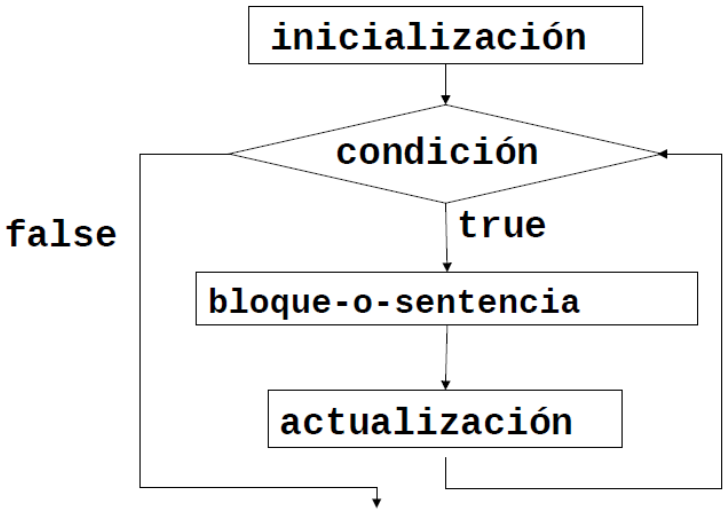
```
do {
   <bloque-o-sentencia>

}while (<condición>);
```

Por ejemplo:

```
import java.util.Scanner;
public class Rango {
	public static void main(String[] args) {
		Scanner teclado = new Scanner(System.in);
		int numero;
		do{
			System.out.println("Introduce número (0-10):");
			numero = teclado.nextInt();
		}while((numero>=0)&&(numero>=10));
		System.out.println("Correcto");
		teclado.close();
	}
}
```

For

Sintaxis java:
for (<**inicialización**>;<**condición**>;<**actualización**>)
<bloque-o-sentencia>



Modo de funcionamiento:

a) Crea e inicializa las variables definidas en **inicialización**.

b) Ejecuta la **actualización** (modifica las variables)

c) En **inicialización** y **actualización** se pueden poner varias variables separadas por comas

d) **inicialización** y **actualización** no tienen por qué compartir variables, aunque es lo habitual (lo útil)

e) Comprueba si se cumple la **condición**, si no se cumple no lo hace ninguna vez (0 a n).

f) Ejecuta el cuerpo del bucle (bloque-o-sentencia) una vez.

g) Comprueba la condición otra vez [vuelve a b)] y así indefinidamente hasta que deje de cumplirse la condición.

h) Si **condición** no existe: bucle infinito.

i) Se considera mala práctica modificar los valores de las variables de control dentro del bucle (para eso está la actualización).

j) Cualquiera de las tres expresiones puede faltar, pero mantenemos los puntos y coma.

Ejemplo For:

```
public class Main {
	public static void main(String[] args) {
		int num,suma_total;
		suma_total=0;
	for (int i=1;i<=15;i++) {
		System.out.print("Introduzca número:
");
		num=Entrada.entero();
		suma_total=suma_total+num;
	}
	System.out.println("La suma total es de:
"+suma_total);
	 }
}
```

Ejemplo FOR anidado

```
public class Main {
	public static void main(String[] args) {
		 int suma;
		for (int i=0;i<4;i++){
			for (int j=3;j>0;j--){
				suma=i*10+j;

	System.out.println(suma);
				}
		}
	}
}
```

Actividad

Practica 2

Escribe un programa que dé como resultado las soluciones x1 y x2 de una ecuación de segundo grado, de la forma: ax2 + bx + c =0

Las soluciones de una ecuación de segundo grado vienen dadas por la fórmula:

$$x_1 = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Las soluciones son reales sólo si b2-4ac es mayor o igual que 0.

Practica 3

Escribe un programa que emplee la sentencia if–else y que imprima como resultado el menor de tres números proporcionados a, b y c.

Practica 4

Escribe un programa que emplee la sentencia else-if para resolver el siguiente planteamiento:

Al efectuar una compra en un cierto almacén, si adquirimos más de 100 unidades de un mismo artículo, nos hacen un descuento de un 40%; entre 25 y 100 un 20%; entre 10 y 24 un 10%; y no hay descuento para una adquisición de menos de 10 unidades.

Se pide calcular el importe a pagar.

Practica 5

Escribe un programa que emplee la sentencia switch que lea una fecha representada por dos enteros, mes, y año, y que dé como resultado los días correspondientes al mes. Esto es:

Introducir mes (##) y año (####): 5 2009

El mes 5 del año 2009 tiene 31 días

Hay que tener en cuenta que febrero puede tener 28 días, o bien 29 si es año bisiesto.

Un año es bisiesto cuando es múltiplo de 4 y no de 100 o cuando es múltiplo de 400.

Practica 6

Escribe un programa que emplee la sentencia while que visualice el código ASCII de cada uno de los caracteres de una cadena de texto introducida por el teclado.

Practica 7

Escribe un programa que emplee la sentencia do … while que calcule la raíz cuadrada de un número n por el método de Newton.

Este método se enuncia así: sea ri la raíz cuadrada aproximada de n. La siguiente raíz aproximada ri +1 se calcula en función de la anterior así:

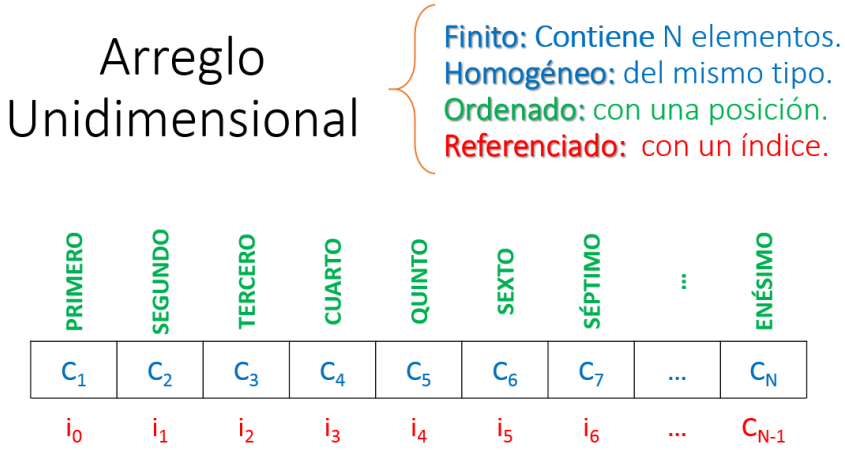
$$r_{i+1} = \frac{\frac{n}{r_i} + r_i}{2}$$

Practica 8

Escribe un programa que emplee la sentencia for que nos pida una cadena de 10 caracteres y nos muestra dicha cadena en forma invertida.

Programación con arreglos y estructuras

Un array es una colección finita de datos del mismo tipo, que se almacenan en posiciones consecutivas de memoria y reciben un nombre común.



Declarar de un array

- En la declaración se crea la referencia al array.
- La referencia será el nombre con el que manejaremos el array en el programa.
- Se debe indicar el nombre del array y el tipo de datos que contendrá.

De forma general un array unidimensional se puede declarar en java de cualquiera de estas dos formas:

1. tipo [] nombreArray;

2. tipo nombreArray[];

tipo: indica el tipo de datos que contendrá. Un array puede contener elementos de tipo básico o referencias a objetos.

nombreArray: es la referencia al array. Es el nombre que se usará en el programa para manejarlo.

Por ejemplo:

int [] edad; //array de datos de tipo int llamado suma

double [] calificaciones; //array de datos de tipo double llamado calificaciones

String [] nombres; //array de datos de tipo String llamado nombres

Inicializar un arreglo:

- Los valores iniciales se escriben entre llaves separados por comas.
- Los valores iniciales deben aparecer en el orden en que serán asignados a los elementos del array.
- El número de valores determina el tamaño del array.

Por ejemplo:

```
double [] calificaciones= {6.7, 7.5, 5.3, 8.75, 3.6, 6.5};
```

Se declara el array notas de tipo double, se reserva memoria para 6 elementos y se les asignan esos valores iniciales.

Ejemplo de códigos

Se declara un array de 7 elementos de tipo double y le asigna valores iniciales.

```
public class arreglo {
	public static void main(String args[]) {
		double[] calificaciones = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7 elementos
		for (int i = 0; i < 7; i++) {
			System.out.print(calificaciones[i] + " "); //se muestra cada elemento del array
		}
	}
}
```

Para evitar errores de acceso al array es recomendable utilizar **length** para recorrer el array completo.

```
public class arreglo {
	public static void main(String args[]) {
		double[] calificaciones = {2.3, 8.5, 3.2, 9.5, 4, 5.5, 7.0}; //array de 7 elementos
		for (int i = 0; i < calificaciones.length; i++) {
			System.out.print(calificaciones[i] + " "); //se muestra cada elemento del array
		}
	}
}
```

Ejemplo de código ingresando datos

```
import java.util.Scanner;
public class arreglo {
	public static void main(String args[]) {
		Scanner entrada = new Scanner (System.in);
		int[] miArreglo = new int[4];
		int suma=0;
		//Se ingresan los valores al arreglo
		for (int i = 0; i < miArreglo.length; i++) {
			System.out.print("Ingrese el número en la posición " + (i) + " : ");
			miArreglo[i] = entrada.nextInt();
		}
		//se realiza el recorrido del arreglo y se suman los varoes
		for (int i = 0; i < miArreglo.length; i++) {
			suma += miArreglo[i];
		}
		System.out.println("El resultado de la suma es "+suma);
	}
```

}

Actividad

Practica 9

Escribe un programa lea 10 números por teclado, los almacene en un array y muestre la media.

Ejercicio 1

Programa que pida 10 calificaciones utilizando arreglo, las sume y calcule el promedio. Al final deberá imprimir el resultado y los valores ingresados.

Ejercicio 2

Crear un programa donde ingresen 10 nombres y 10 calificaciones utilizando arreglos, al final deberán imprimir el nombre con la calificación correspondiente.

Ejercicio 3

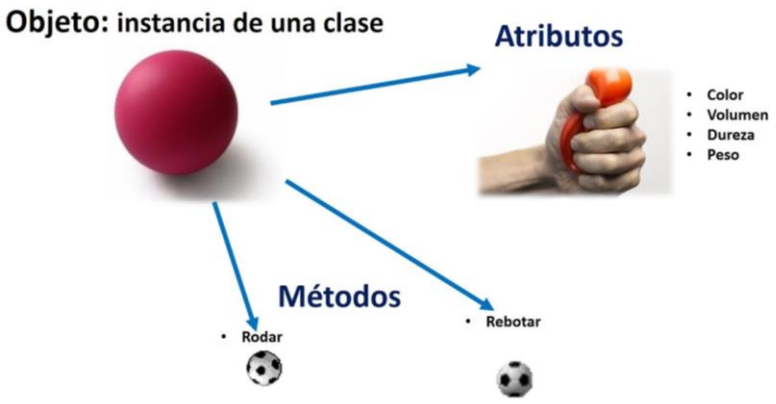
Crear un programa donde ingresen 10 nombres y 10 calificaciones utilizando arreglos, al final deberán imprimir la suma total de calificaciones y el promedio general.

Unidad 1

1.2 Modela y codifica programas de cómputo haciendo uso del paradigma orientado a objetos, a través del uso de sentencias de control, objetos y clases

Actividades según programa de estudios Conalep

Programación de objetos y clases



Objetos

- Los objetos son/representan cosas
- Los objetos pueden ser simples o complejos
- Los objetos pueden ser reales o imaginarios

Atributos

- Valores o características de los objetos
- Permiten definir el estado del objeto u otras cualidades

Por ejemplo:

Auto: {Marca, Color, Potencia, Modelo}

Métodos (u operaciones)

- Los métodos pueden devolver un valor al acabar su ejecución (Valor de retorno)
- Acciones que puede realizar un objeto

Por ejemplo:

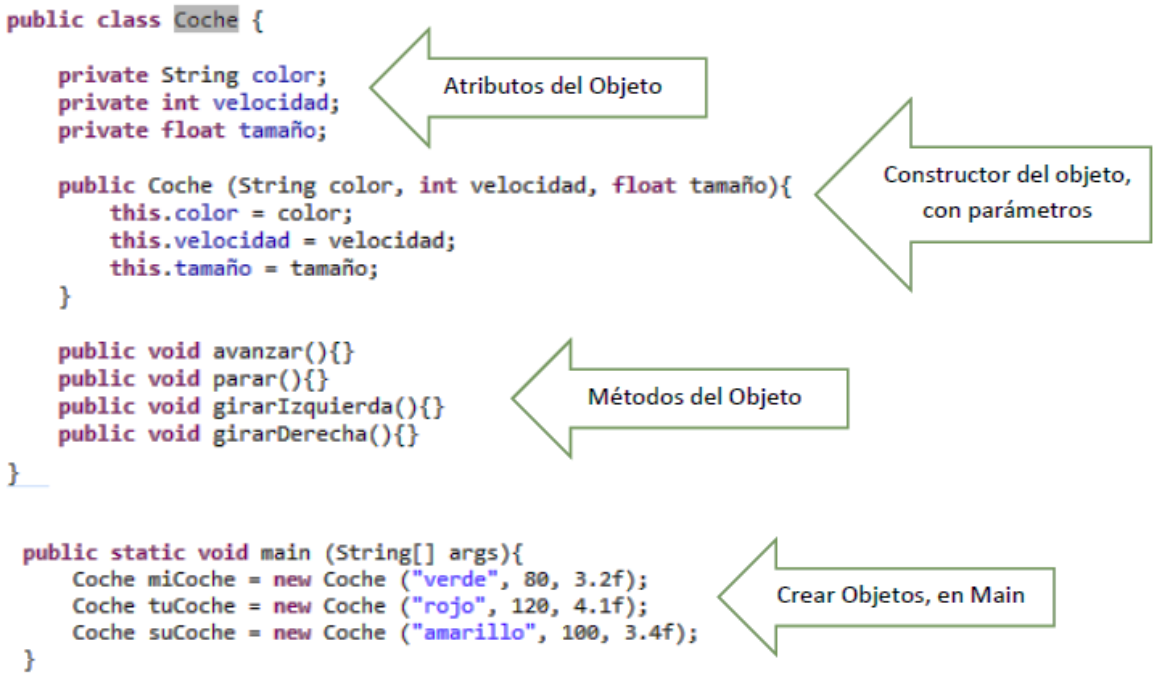
1. Arrancar motor
2. Parar motor
3. Acelerar
4. Frenar
5. Girar a la derecha (grados)
6. Girar a la izquierda (grados)
7. Cambiar marcha (nueva marcha)

Clases

- Representan un tipo particular de objetos
- Objetos con características y comportamiento similar
- Categorías de objetos
- Cada clase tiene asociado un código (definición de la clase), que determina

- Los atributos que tienen los objetos de la clase
- Los métodos que pueden ejecutar los objetos de la clase y cómo lo hacen

Programar orientado a objetos consiste en escribir código de clases de objetos, por ejemplo:



Actividad

1. Elabora un programa que represente la clase Alumno, creando al menos 3 objetos.

2. Crea una clase que se llame Figura, donde tenga los datos de base, altura, color, forma.

Deberás crear:

1. Constructor con todos los parámetros.
2. Constructor con los siguientes parámetros: nombre, base, altura.
3. Constructor con los parámetros: nombre, color y forma.

En el método main deberás crear al menos 2 objetos por cada constructor declarado.

Unidad 2

2.1 Elabora aplicaciones mediante la interacción de los objetos y actores del sistema.

Actividades según programa de estudios Conalep

Métodos get y set

Convenio para acceder a atributos con visibilidad privada son:

- **Método Get** Devuelve el valor de una variable
- **Método Set** Modifica el valor de una variable

Ejemplo de código

```
public class Empleado {
  //Atributos
   private String nombre;
   private String apellido;
   private int edad;
   private double salario;

   //Metodos publicos
   //Devuelve el nombre del empleado
   public String getNombre() {
      return nombre;
   }
   //Modifica el nombre de un empleado
   public void setNombre(String nombre) {
      this.nombre = nombre;
   }
   public int getEdad() {
      return edad;
   }
   public void setEdad(int edad) {
      this.edad = edad;
   }
   public double getSalario() {
      return salario;
   }
   public boolean plus (double sueldoPlus){
      boolean aumento=false;
      if (edad>40 && compruebaNombre()){
         salario+=sueldoPlus;
         aumento=true;
      }
      return aumento;
   }
   //Metodos privados
   private boolean compruebaNombre(){
```

```
        if(nombre.equals("")){
            return false;
        }
        return true;
    }
   //Constructor por defecto
        public Empleado(){
        this.nombre="";
        this.apellido="";
        this.edad=0;
        this.salario=0;
    }
    public Empleado(String nombre, String apellido, int edad, double salario){
        this.nombre=nombre;
        this.apellido=apellido;
        this.edad=edad;
        this.salario=salario;
    }
//Programa principal para utilizar los métodos GET, SET y Constructores
    public static void main(String[] args) {
        Empleado empleado1=new Empleado ("Fernando", "Ureña", 23, 800);
        Empleado empleado2=new Empleado ("", "Lopez", 50 ,1800);
        //Mostramos el valor actual del salario del empleado1
        System.out.println("El salario del empleado1 es "+empleado1.getSalario());
        //Modificamos la edad del empleado1
        empleado1.setEdad(43);
        empleado1.plus(100);
        //Mostramos el salario de nuevo, ahora tendra 100 mas
        System.out.println("El salario actual del empleado1 es "+empleado1.getSalario());
        //Modificamos el nombre del empleado2
        empleado2.setNombre("Antonio");
        empleado2.plus(100);
        //Mostramos el salario de nuevo, ahora tendra 100 mas
        System.out.println("El salario del empleado2 es "+empleado2.getSalario());
    }
}
```

Herencia

- Es un mecanismo que permite la definición de una clase a partir de la definición de otra ya existente.
- Permite compartir automáticamente métodos y datos entre clases, subclases y objetos.
- Está fuertemente ligada a la reutilización del código en la POO. Esto es, el código de cualquiera de las clases puede ser utilizado sin más que crear una clase derivada de ella, o bien una subclase.

Hay dos tipos de herencia: **Herencia Simple** y **Herencia Múltiple**. La primera indica que se pueden definir nuevas clases solamente a partir de una clase inicial mientras que la segunda indica que se pueden definir nuevas clases a partir de dos o más clases iniciales.

Ventajas de la Herencia

Entre las principales ventajas que ofrece la herencia en el desarrollo de aplicaciones, están:

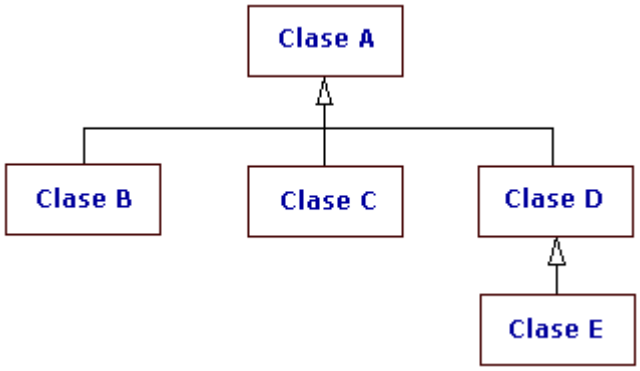
- Reutilización del código: En aquellos casos donde se necesita crear una clase que, además de otros propios, deba incluir los métodos definidos en otra, la herencia evita tener que reescribir todos esos métodos en la nueva clase.
- Mantenimiento de aplicaciones existentes: Utilizando la herencia, si tenemos una clase con una determinada funcionalidad y tenemos la necesidad de ampliar dicha funcionalidad, no necesitamos modificar la clase existente (la cual se puede seguir utilizando para el tipo de programa para la que fue diseñada) sino que podemos crear una clase que herede a la primera, adquiriendo toda su funcionalidad y añadiendo la suya propia.

Superclase y Subclases

El concepto de herencia conduce a una estructura jerárquica de clases o estructura de árbol, lo cual significa que en la POO todas las relaciones entre clases deben ajustarse a dicha estructura.

En esta estructura jerárquica, cada clase tiene sólo una clase **padre**. La clase padre de cualquier clase es conocida como su superclase. La clase hija de una superclase es llamada una subclase.

- La palabra clave utilizada para la herencia es **extends**
- Una superclase puede tener cualquier número de subclases.
- Una subclase puede tener sólo una superclase.



- A es la superclase de B, C y D.
- D es la superclase de E.
- B, C y D son subclases de A.
- E es una subclase de D

Ejemplo de cómo usar la herencia

```
//Clase para objetos de dos dimensiones
class Figura {
   double base;
   double altura;

   void mostrarDimension(){
      System.out.println("La base y altura es: "+base+" y "+altura);
   }
}

//Una subclase de Figura para Triangulo
class Triangulo extends Figura {
   String estilo;
   double area(){
      return base*altura/2;
   }
   void mostrarEstilo(){
      System.out.println("Triangulo es: "+estilo);
   }
}

class Lados3{
   public static void main(String[] args) {
      Triangulo t1=new Triangulo();
      Triangulo t2=new Triangulo();

      t1.base=4.0;
      t1.altura=4.0;
      t1.estilo="Estilo 1";

      t2.base=8.0;
      t2.altura=12.0;
      t2.estilo="Estilo 2";

      System.out.println("Información para T1: ");
      t1.mostrarEstilo();
      t1.mostrarDimension();
      System.out.println("Su área es: "+t1.area());

      System.out.println();

      System.out.println("Información para T2: ");
      t2.mostrarEstilo();
      t2.mostrarDimension();
      System.out.println("Su área es: "+t2.area());

   }
}
```

Actividad

Practica 12

Cree una clase Rectángulo con los atributos longitud y ancho, cada uno con un valor predeterminado igual a 1. Proporcione funciones miembro que calculen el perímetro y el área del rectángulo.
Además, proporcione las funciones set (establecer) y get (obtener) para los atributos longitud y ancho respectivamente.
Las funciones set (establecer) deben verificar que longitud y ancho contengan números reales mayores que cero y menores que veinte, de lo contrario mostraran el valor de cero.
Escriba un método main cuya ejecución muestre el siguiente resultado en pantalla:

```
Creamos un rectangulo de 2 de alto por 3.2 de ancho
La altura es: 2
El ancho es: 3.2
El area es: 6.4
El perimetro es: 10.4

Se intenta introducir un valor mayor 20 en el ancho
La altura es: 2
El ancho es: 0
El area es: 0
El perimetro es: 4
```

Practica 15

En una empresa automotriz se tienen 3 tipos de empleados: administrativos, mecánicos y vendedores. En general, para todos los empleados se tiene los datos RFC (Registro Federal de Contribuyentes), el nombre, el departamento y el puesto. En particular; para el empleado administrativo se tiene el dato sueldo mensual; para el mecánico se tiene el precio del trabajo, tantas veces como trabajos haya realizado; y para el vendedor se tiene el precio del auto, por cada auto que vendió.
El sueldo quincenal se calcula:

- · Para el administrativo, sueldo mensual entre 2.
- · Para el mecánico, el 4% del valor total.
- · Para el vendedor, el salario mínimo, más el 2 por ciento del valor de la venta realizada.

La idea es que se use una superclase Empleado que contendrá los datos RFC, el nombre, el departamento, y el puesto; y los métodos para establecer y obtener cada uno de los datos. De esa superclase derivar tres subclases: EmpAdmvo, EmpMecánico y EmpVendedor; en cada una de las cuales se heredarán los datos y los métodos de la superclase; además, cada subclase de estas, deberán tener sus propios datos y métodos para establecer los datos necesarios, calcular el sueldo quincenal correspondiente y obtenerlo para imprimirlo. Asimismo, deberá haber una clase controlador que permita leer los datos y utilice el modelo para representar y solucionar el problema.

Unidad 2

2.1 Elabora aplicaciones mediante la interacción de los objetos y actores del sistema.

Actividades según programa de estudios Conalep

Uso de sobrecarga de operadores.

Operadores Unarios

Los operadores unarios en Java son aquellos que solo requieren un operando para funcionar.

Los operadores unitarios que tenemos en Java son:

Operador	Descripción
+	Operador unario suma. Indica un número positivo.
-	Operador unario resta. Niega una expresión.
++	Operador de incremento. Incrementa el valor en 1.
–	Operador de decremento. Decrementa el valor en 1.
!	Operador de complemento lógico. Invierte el valor de un booleano

Operadores unarios suma o resta

Los operadores unitarios de suma o resta son muy sencillos de utilizar. En el caso del operador unitario suma su uso es redundante. Con el operador unitario resta podemos invertir un valor.

Por ejemplo podríamos tener el siguiente código:

```
int valor = 2;
System.out.println(-valor); // Imprimirá por pantalla un -2
```

Operadores de incremento y decremento

Los operadores de incremento se pueden aplicar como prefijo o como sufijo.

```
++ variable;
variable ++;
-- variable;
variable --;
```

En todos los casos el valor de la variable acabará con una unidad más (para el operador de incremento) o con una unidad menos (para el operador de decremento).

Por ejemplo:

```
public class operators {
  public static void main(String[] args)   {
    int a = 20, b = 10, c = 0, d = 20, e = 40, f = 30;
    boolean condition = true;
    // operador de pre-incremento
    // a = a+1 y entonces c = a;
    c = ++a;
    System.out.println("Valor de c (++a) = " + c);

    // operador de post-incremento
    // c=b entonces b=b+1 (b pasa a ser 11)
    c = b++;
```

```
    System.out.println("Valor de c (b++) = " + c);

    // operador de pre-decremento
    // d=d-1 entonces c=d
    c = --d;
    System.out.println("Valor de c (--d) = " + c);

    // operador de post-decremento
    // c=e entonces e=e-1 (e pasa a ser 39)
    c = e--;
    System.out.println("Valor de c (e--) = " + c);

    // Operador lógico not
    System.out.println("Valor de !condition = " + !condition);

  }
}
```

Operador de asignación (=)

El operador de asignación se usa para asignar un valor a cualquier variable. Tiene una asociación de derecha a izquierda, es decir, el valor dado en el lado derecho del operador se asigna a la variable de la izquierda y, por lo tanto, el valor del lado derecho debe declararse antes de usarlo o debe ser una constante.

El formato general del operador de asignación es, variable = valor;

+ =, para sumar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

− =, para restar el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

*** =**, para multiplicar el operando izquierdo con el operando derecho y luego asignándolo a la variable de la izquierda.

/ =, para dividir el operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda.

^ =, para aumentar la potencia del operando izquierdo al operando derecho y asignarlo a la variable de la izquierda.

% =, para asignar el módulo del operando izquierdo con el operando derecho y luego asignarlo a la variable de la izquierda

Por ejemplo

```
public class operators {
    public static void main(String[] args)     {
        int a = 20, b = 10, c, d, e = 10, f = 4, g = 9;
        // operador de asignación simple
        c = b;
        System.out.println("Valor de c = " + c);

        // Esta siguiente declaración arrojaría una exception
        // porque el valor del operando derecho debe ser inicializado
        // antes de la asignación, entonces el programa no
        // compila.
        // c = d;
```

```
        // operadores de asignación simples
        a = a + 1;
        b = b - 1;
        e = e * 2;
        f = f / 2;
        System.out.println("a,b,e,f = " + a + "," + b + "," + e + "," + f);
        a = a - 1;
        b = b + 1;
        e = e / 2;
        f = f * 2;
        // operados de asignación compuestos/cortos
        a += 1;
        b -= 1;
        e *= 2;
        f /= 2;
        System.out.println("a,b,e,f (usando operadores cortos)= " + a + "," + b + "," + e + "," + f);
    }
}
```

Operador de Instancia (instanceof)

El operador de instancia se usa para verificar el tipo. Se puede usar para probar si un objeto es una instancia de una clase, una subclase o una interfaz.

Formato general: objecto instanceof class/subclass/interface

Por ejemplo:

```
class operators {
    public static void main(String[] args)     {
        Person obj1 = new Person();
        Person obj2 = new Boy();
        // Como obj1 es de tipo Person, no es una
        // instancia de Boy o interfaz
        System.out.println("obj1 instanceof Person: " + (obj1 instanceof Person));
        System.out.println("obj1 instanceof Boy: " + (obj1 instanceof Boy));
        System.out.println("obj1 instanceof MyInterface: " + (obj1 instanceof MyInterface));
        // Dado que obj2 es de tipo Boy, cuya clase padre es
        // Person e implementa la interfaz Myinterface
        // es una instancia de todas estas clases
        System.out.println("obj2 instanceof Person: " + (obj2 instanceof Person));
        System.out.println("obj2 instanceof Boy: " + (obj2 instanceof Boy));
        System.out.println("obj2 instanceof MyInterface: " + (obj2 instanceof MyInterface));
    }
}
 class Person { }
 class Boy extends Person implements MyInterface { }
 interface MyInterface { }
```

Operadores shift

Los operadores shift de Java se usan para desplazar los bits de un número hacia la izquierda o hacia la derecha, multiplicando o dividiendo el número por dos, respectivamente. Se pueden usar cuando tenemos que multiplicar o dividir un número por dos.

<<, operador de desplazamiento a la izquierda: desplaza los bits del número hacia la izquierda y llena con "0" los bits desplazados. Efecto similar a la multiplicación del número con una potencia de dos.

>>, Operador de desplazamiento a la derecha firmado: desplaza los bits del número a la derecha y llena con "0" los bits desplazados. El bit más a la izquierda depende del signo del número inicial. Efecto similar a partir de dividir el número con alguna potencia de dos.

>>>, Operador de cambio a la derecha sin signo: desplaza los bits del número a la derecha y llena con "0" los bits desplazados. El bit más a la izquierda se establece en 0.

Ejemplo

```
public class operators{
   public static void main(String[] args)    {
      int a = 0x0005;
      int b = -10;
      // operador de desplazamiento a la izquierda
      // 0000 0101<<2 =0001 0100(20)
      // similar a 5*(2^2)
      System.out.println("a<<2 = " + (a << 2));
      // operador de desplazamiento a la derecha
      // 0000 0101 >> 2 =0000 0001(1)
      // similar a 5/(2^2)
      System.out.println("a>>2 = " + (a >> 2));
      // operador de cambio a la derecha sin firmar
      System.out.println("b>>>2 = "+ (b >>> 2));
   }
}
```

Actividad

Practica 16

Escribir un programa que permita realizar la suma y la resta de números complejos. Un número complejo estará definido por la clase CComplejo y para realizar las operaciones solicitadas esta clase incluirá un método para sobrecargar el operador + y otro para el -.

Practica 17

De acuerdo con el código de la practica 16 realiza las siguientes operaciones:

1.1

a) Crea la función miembro "esIgual", que devuelve un valor del tipo int, que nos dará falso o verdadero (0 o 1) dependiendo de que los dos números complejos evaluados sean distintos o iguales.

b) Crear dos funciones miembros, que incrementen un número complejo en una unidad, tanto en la parteReal, como en la parteImaginaria, en dos versiones "preImcremento" (++X) y "posIncremento" (X++) (se incrementa el número complejo después de ser utilizado).

1.2

a) Sobrecargar el operador +, eliminando la función miembro "suma" (y opcionalmente el operador -, eliminando la función miembro “resta”).

b) Sobrecargar el operador ==, eliminando la función miembro "esIgual" (y opcionalmente el operador !=).

c) Sobrecargar el operador ++ (como preIncremento y como posIncremento), eliminando las funciones miembro "preIncremento" y "posIncremento" (y opcionalmente el operador – en las mismas condiciones).

d) Sobrecargar el operador <<, eliminando la función miembro "imprime" (y opcionalmente el operador >>).

Practica 18

Genere una clase CRacional (fracciones) con las siguientes capacidades:

a) Cree un constructor para prevenir un denominador 0 en una fracción, reduzca o simplifique las fracciones que no se encuentren en forma reducida y evite denominadores negativos.

b) Sobrecargue los operadores de suma y resta para esta clase. (Opcional multiplicación y división)

c) Sobrecargue el operador de igualdad (==) para esta clase. (Opcional (!=))

d) Sobrecargue el operador (++) como prefijo y sufijo. (Opcional (--))

e) Sobrecargue los operadores (<<) y (>>).

Unidad 2

2.1 Elabora aplicaciones mediante la interacción de los objetos y actores del sistema.

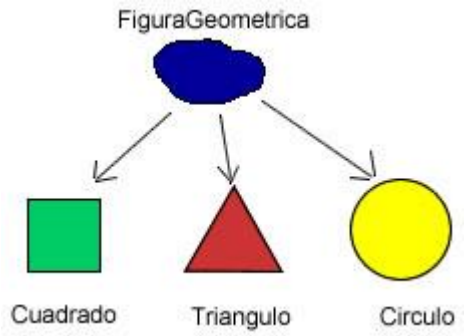
Actividades según programa de estudios Conalep

Polimorfismo

Polimorfismo se refiere a la habilidad para aparecer en varias formas.

Polimorfismo en programas Java significa:

- La habilidad de una variable referencia para cambiar su comportamiento de acuerdo con la instancia del objeto que contiene
- Esto permite que múltiples objetos de diferentes subclases sea tratados como objetos de una superclase única, mientras que automáticamente se selecciona los métodos apropiados a aplicar a un objeto en particular basado en la subclase a la que pertenece



Beneficios del Polimorfismo

Simplicidad

- Si se necesita escribir código que trata con una familia de subtipos, el código puede ignorar los detalles específicos de tipo y sólo interactuar con el tipo base de la familia
- Aun cuando el código piense que está usando un objeto de la clase base, la clase del objeto podría ser la clase base o cualquiera de sus subclases

Extensibilidad

- Se pueden añadir subclases posteriormente a la familia de tipos, y los objetos de estas nuevas subclases podría trabajar con el código existente

Tres formas de polimorfismo en un programa Java

1. **Overriding de método.** Métodos de una subclase sobrescriben los métodos de una superclase
2. **Overriding de método (implementación) de los métodos abstractos.** Métodos de una superclase implementa los métodos abstractos de una clase abstracta.
3. **Overriding de método (implementación) a través de interface.** Métodos de una superclase implementa los métodos abstractos de la interface

Por ejemplo, dada una clase base forma, el polimorfismo permite al programador definir diferentes métodos area para cualquier número de clases derivadas, tales, como circulo, rectángulo, y triangulo

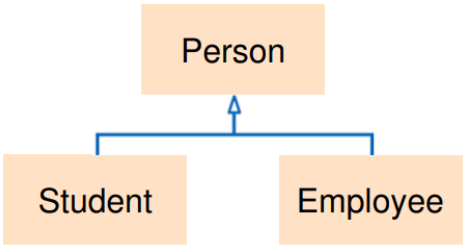
– El método área de circulo, rectángulo, y triangulo se implementa de manera diferente

• No importa qué forma tiene un objeto, aplicándole el método área devolverá el resultado correcto

Ejemplo 1: polimorfismo

Dada la clase padre Person y la clase hija Student, se añade otra subclase Person que es Employee

La gráfica de la jerarquía de clases es:



Se puede crear una referencia que es del tipo de la superclase, Person, hacia un objeto de su subclase Student

```
public static main( String[] args ) {
        Student studentObject = new Student();
        Employee employeeObject = new Employee();
        Person ref = studentObject; //referencia Person
        // al objeto Student
        // Llamada de getName() de la instancia de objeto Student
        String name = ref.getName();
}
```

Supongamos ahora que hay un método getName en la superclase Person y que este método es sobrescrito en las subclases Student y Employee

```
public class Student {
        public String getName(){
                System.out.println("Nombre estudiante:" + name);
                public class Student { public String getName(){
                        System.out.println("Nombre estudiante:" + name);
                        System.out.println("Nombre estudiante:" + name);
                        return name;
                }
        }
        public class Employee { public String getName(){
                System.out.println("Nombre empleado:" + name);
                return name;
        }
}
```

Volviendo al método main, cuando se invoca el método getName de la referencia Person ref, el método getName del objeto Student será llamado

• Si se asigna ref a un objeto Employee, el método getName de Employee será llamado

```
public static main (String[] args) {
        Student studentObject = new Student();
        Employee employeeObject = new Employee();
        //ref apunta a un objeto Student
        Person ref = studentObject;
        //metodo getName() de la clase Student es invocado
        public static main (String[] args) {
                Student studentObject = new Student();
```

```
                Employee employeeObject = new Employee();
                //ref apunta a un objeto Student
                Person ref = studentObject;
                //metodo getName() de la clase Student es invocado
                //metodo getName() de la clase Student es invocado
                String temp= ref.getName();
                System.out.println(temp);
                //ref apunta ahora a un objeto Employee
                ref = employeeObject;
                //metodo getName() de la clase Employee es invocado
                String temp = ref.getName();
                System.out.println(temp);
        }
}
```

Actividad

Practica 19

Escribe un programa que ofrezca un menú de opciones, mediante el cual se pueda seleccionar calcular el volumen de las figuras geométricas: cubo, cilindro, cono y esfera. Una vez seleccionada la opción, que permita solicitar y leer el nombre de la figura y los datos necesarios para calcular el volumen correspondiente, imprimir el nombre de la figura y el volumen.

Volumen de cubo = $Arista^3$

Volumen de cilindro = $\pi\ r^2\ h$

Volumen de cono = $1 / 3\ \pi\ r^2\ h$

Volumen de esfera = $4 / 3\ \pi\ r^3$

La idea es que se use una superclase abstracta Figura que contendrá el dato nombre y los métodos para establecerlo y obtenerlo; además el dato volumen, un método abstracto para calcular el volumen y un método para obtenerlo e imprimirlo. De esa superclase derivar cuatro subclases: Cubo, Cilindro, Cono y Esfera; en cada una de las cuales se heredarán los datos y los métodos de la superclase Figura. Cada subclase de éstas, deberá tener sus propios datos y los métodos necesarios para establecerlos; además del método calcular el volumen de la figura correspondiente. En virtud de que calcular volumen es un método abstracto heredado de la superficie abstracta Figura, cada una de las subclases derivadas, lo deberá implementar de acuerdo con la forma que le corresponda; aplicando el polimorfismo.

Practica 20

Crear una clase base abstracta vehículo que contenga una función virtual para mostrar los atributos de un objeto en pantalla y dos clases derivadas que deberán concretar la clase anterior: vehículo Terrestre y vehículo Aéreo.

La clase base definirá dos atributos: costo y año del vehículo y dos funciones para obtener sus valores. Las clases derivadas vehículo Terrestre y vehículo Aéreo contienen los atributos kilometraje y horas de vuelo respectivamente. Además, deberán ofrecer funciones para obtener y establecer esos atributos.

Escribir una función de prueba main() que cree objetos de las clases derivadas y un apuntador de clase base que haga uso de la función virtual que se ha concretado en cada una de las clases derivadas.

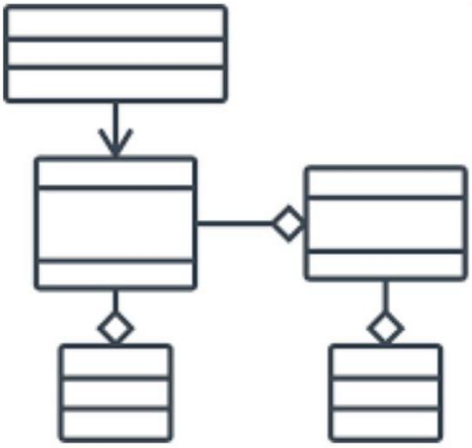
Unidad 2

2.2 Asegura la integridad de los datos implementando medidas de seguridad haciendo uso de patrones de diseño ya sea para la autenticación como para él cifrado de los datos.

Actividades según programa de estudios Conalep

Identificación de patrones de diseño para la autenticación en aplicaciones

¿Qué es un patrón de diseño?



Christopher Alexander dice: *"Cada patrón describe un problema que ocurre una y otra vez en nuestro entorno, para describir después el núcleo de la solución a ese problema, de tal manera que esa solución pueda ser usada más de un millón de veces, sin hacerlo ni siquiera dos veces de la misma forma".*

En general, un patrón tiene cuatro elementos esenciales:

1. **El nombre del patrón** se utiliza para describir un problema de diseño, su solución, y consecuencias en una o dos palabras. Nombrar un patrón incrementa inmediatamente nuestro vocabulario de diseño. Esto nos permite diseños a un alto nivel de abstracción. Tener un vocabulario de patrones nos permite hablar sobre ellos con nuestros amigos, en nuestra documentación, e incluso a nosotros mismos.
2. **El problema describe cuando aplicar el patrón**. Se explica el problema y su contexto. Esto podría describir problemas de diseño específicos tales como algoritmos como objetos. Podría describir estructuras de clases o objetos que son sintomáticas de un diseño inflexible. Algunas veces el problema incluirá una lista de condiciones que deben cumplirse para poder aplicar el patrón.
3. **La solución** describe los elementos que forma el diseño, sus relaciones, responsabilidades y colaboraciones. La solución no describe un diseño particular implementación, porque un patrón es como una plantilla que puede ser aplicada en diferentes situaciones. En cambio, los patrones proveen una descripción abstracta de un problema de diseño y como una disposición general de los elementos (clases y objetos en nuestro caso) lo soluciona.
4. **Las consecuencias** son los resultados de aplicar el patrón. Estas son muy importantes para la evaluación de diseños alternativos y para comprender los costes y beneficios de la aplicación del patrón.

Descripción de patrones de diseño

1. Nombre del patrón

Esta sección consiste de un nombre del patrón y una referencia bibliografía que indica de donde procede el patrón. El nombre es significativo y corto, fácil de recordar y asociar a la información que sigue.

2. Objetivo

Esta sección contiene unas pocas frases describiendo el patrón. El objetivo aporta la esencia de la solución que es proporcionada por el patrón. El objetivo esta dirigido a programadores con experiencia que pueden reconocer el patrón como uno que ellos ya conocen, pero para el cual ellos no le han dado un nombre. Después de reconocer el patrón por su nombre y objetivo, esto podría ser suficiente para comprender el resto de la descripción del patrón.

3. Contexto

La sección de Contexto describe el problema que el patrón soluciona. Este problema suele ser introducido en términos de un ejemplo concreto. Después de presentar el problema en el ejemplo, la sección de Contexto sugiere una solución de diseño a ese problema.

4. Aplicabilidad

La sección Aplicabilidad resume las consideraciones que guían a la solución general presentada en la sección Solución. En que situaciones es aplicable el

patrón.

5. Solución

La sección Solución es el núcleo del patrón. Se describe una solución general al problema que el patrón soluciona. Esta descripción puede incluir, diagramas y

texto que identifique la estructura del patrón, sus participantes y sus colaboraciones para mostrar como se soluciona el problema. Debe describir tanto la estructura dinámica como el comportamiento estático.

6. Consecuencias

La sección Consecuencias explica las implicaciones, buenas y malas, del uso de la solución.

7. Implementación

La sección de Implementación describe las consideraciones importantes que se han de tener en cuenta cuando se codifica la solución. También puede contener algunas variaciones o simplificaciones de la solución.

8. Usos en el API de Java

Cuando hay un ejemplo apropiado del patrón en el núcleo del API de Java es comentado en esta sección. Los patrones que no se utilizan en el núcleo del API de Java no contienen esta sección.

9. Código del ejemplo

Esta sección contiene el código del ejemplo que enseña una muestra de la implementación para un diseño que utiliza el patrón. En la mayoría de estos casos, este será el diseño descrito en la sección de Contexto.

10. Patrones relacionados

Esta sección contiene una lista de los patrones que están relacionados con el patrón que se describe.

Clasificación de los patrones de diseño

Fundamentales	De creación	De Partición	Estructurales	De comportamiento	De Concurrencia
Delegation (When Not to Use Inheritance)	Factory Method	Layered Initialization	Adapter	Chain of Responsibility	Single Threaded Execution
Interface	Abstract Factory	Filter	Iterator	Command	Guarded Suspension
Immutable	Builder	Composite	Bridge	Little Language	Balking
Marker Interface	Prototype		Facade	Mediator	Scheduler
Proxy	Singleton		Flyweight	Snapshot	Read/Write Lock
	Object Pool		Dynamic Linkage	Observer	Producer-Consumer
			Virtual Proxy	State	Two-Phase Termination
			Decorator	Null Object	
			Cache Management	Strategy	
				Template Method	
				Visitor	

Patrones de Creación

Los patrones de creación muestran la guía de cómo crear objetos cuando sus creaciones requieren tomar decisiones. Estas decisiones normalmente serán resueltas dinámicamente decidiendo que clases instanciar o sobre que objetos un objeto delegará responsabilidades. La valía de los patrones de creación nos dice como estructurar y encapsular estas decisiones.

Factory Method

Tu escribes una clase para reutilizar con tipos arbitrarios de datos. Tu organizas esta clase de tal forma que se puedan instanciar otras clases sin depender de ninguna de las clases que puedan ser instanciadas. La clase reutilizable es capaz de permanecer independiente de las otras clases instanciadas delegando la elección de que clase instanciar a otros objetos y referirse a los objetos recién creados a través de un interface común. Define una interface para crear un objeto, dejando a las subclases decidir el tipo específico. Permite delegar la responsabilidad de instanciación a las subclases.

Abstract Factory

Dado un conjunto de clases abstractas relacionadas, el patrón Abstract Factory permite el modo de crear instancias de estas clases abstractas desde el correspondiente conjunto de subclases concretas. Proporciona una interfaz para crear familias de objetos relacionados o dependientes sin especificar su clase concreta. El patrón

Abstract Factory puede ser muy útil para permitir a un programa trabajar con una variedad compleja de entidades externas, tales como diferentes sistemas de ventanas con una funcionalidad similar.

Builder

Permite a un objeto cliente construir un objeto complejo especificando solamente su tipo y contenido. El cliente es privado de los detalles de construcción del objeto. Separa la construcción de un objeto complejo de su representación, para que el mismo proceso de construcción permita crear varias representaciones.

Prototype

Permite a un objeto crear objetos personalizados sin conocer su clase exacta o los detalles de cómo crearlos. Su tarea es dar objetos prototipo a un objeto que inicializa la creación de objetos. El objeto de creación e inicialización crea objetos mandando a sus objetos prototipo que hagan una copia de si mismos.

Singleton

Garantiza que solamente se crea una instancia de la clase y provee un punto de acceso global a él. Todos los objetos que utilizan una instancia de esa clase usan la misma instancia.

Object Pool

Administra la reutilización de objetos cuando un tipo de objetos es caro de crear o solamente un número limitado de objetos puede ser creado.

Actividad

1. Elabora un mapa conceptual o cuadro sinoptico de la lectura anterior.

2. Representa en Diagrama de clase, cual es la solución para cada uno de los patrones de creación

3. Diseña las pantallas para la Solicitud de usuario y contraseña en un sistema.

Unidad 2

2.2 Asegura la integridad de los datos implementando medidas de seguridad haciendo uso de patrones de diseño ya sea para la autenticación como para él cifrado de los datos.

Actividades según programa de estudios Conalep

Manejo de Excepciones

Una excepción es un evento que ocurre durante la ejecución de un programa que rompe el flujo normal de ejecución. Cuando se habla de excepciones nos referimos a un evento excepcional.

Cuando se produce una excepción dentro de un método, se crea un objeto que contiene información sobre la excepción y retorna en forma inusual al código llamador con la información de la excepción.

- La rutina receptora de la excepción es responsable de reaccionar a tal evento inesperado.
- Cuando creamos un objeto para la excepción y lo pasamos al código llamador decimos que lanzamos una excepción (Throw an exception)
- Si el método llamador no tiene un manejador de la excepción se busca hacia atrás en la pila de llamados anidados hasta encontrarlo.
- Decimos que el manejador atrapa la excepción (palabra reservada "catch")

Propaga los errores hacia atrás en la secuencia de llamados anidados.

- Se agrupan los errores según su naturaleza.
    - Si hay más de un archivo que se abre, basta con un código para capturar tal caso.
    - Si se lanzan excepciones que son todas subclases de una base, basta con capturar la base para manejar cualquiera de sus instancias derivadas.
- En Java los objetos lanzados deben ser instancias de clases derivadas de Throwable.

```
public class TryCatchFinally {

    public static void main(String[] args) {

        try {
            String mensajes[] = {"Primero", "Segundo",  "Tercero" };
            for (int i=0; i<=3; i++)
                System.out.println(mensajes[i]);

        } catch ( ArrayIndexOutOfBoundsException e ) {
            System.out.println("Error: apuntador fuera del rango del arreglo.");
        }
    }
}
```

- Si una excepción no es procesada, debe ser relanzada.

Tipos de Excepciones

Las hay de dos tipos

1. ***Aquellas generadas por el lenguaje Java***. Éstas se generan cuando hay errores de ejecución, como al tratar de acceder a métodos de una referencia no asignada a un objeto, división por cero, etc.
2. ***Aquellas no generadas por el lenguaje***, sino incluidas por el programador.

El compilador chequea por la captura de las excepciones lanzadas por los objetos usados en el código. Si una excepción no es capturada, debe ser relanzada

Captura de Excepciones

La palabra reservada **finally** permite definir un tercer bloque de código dentro del manejador de excepciones. Este bloque le indica al programa las instrucciones a ejecutar de manera independiente de los bloques try-catch, es decir, si el código del bloque try se ejecuta de manera correcta, entra al bloque finally; si se genera un error, después de ejecutar el código del bloque catch ejecuta el código del bloque finally.

```
public class TryCatchFinally {

    public static void main(String[] args) {

    try {
        float equis = 5/0;
        System.out.println("Equis = " + equis);
    } catch ( ArithmeticException e ) {
        System.out.println("Error: división entre cero.");

    } finally {
        System.out.println("A pesar de todo, se ejecuta el bloque finally.");
    }
  }
}
```

Código sin generar Excepción

```
public class TryCatchFinally {

    public static void main(String[] args) {

    try {
        float equis = 5/2;
        System.out.println("Equis = " + equis);
    } catch ( ArithmeticException e ) {
        System.out.println("Error: división entre cero.");

    } finally {
        System.out.println("A pesar de todo, se ejecuta el bloque finally.");
    }
  }
}
```

El **bloque catch** permite capturar excepciones, es decir, manejar los errores que genere el código del bloque try en tiempo de ejecución impidiendo así que el programa deje de ejecutarse y posibilitando al desarrollador enviar el error generado.

Es posible tener más de un bloque catch dentro del manejador de excepciones, pero cuidando el orden de captura, es decir, las excepciones se deben acomodar de las más específicas a las más generales, como se muestra a continuación:

- catch (ClassNotFoundException e) {...}
- catch (IOException e) {...}
- catch (Exception e) {...}

Throws

Existen métodos que obligan a ejecutarse dentro de un manejador de excepciones (es decir, son marcadas), debido a que el método define que va a lanzar una excepción.

Para indicar que un método puede lanzar una excepción se utiliza la palabra reservada throws seguida de la o las excepciones que puede arrojar dicho método. La sintaxis es la siguiente:

[modificadores] valorRetorno nombreMetodo() throws Excepcion1, Excepcion2 {

    // Bloque de código del método

}

Por ejemplo

```
public class PropagaExcepcion {

    public static int miMetodo(int a, int b) throws ArithmeticException{
        if(b == 0){
            throw new ArithmeticException();
        }
        int c =a / b;
        return c;
    }

    public static void main(String[] args) {
        try{
            int division= miMetodo(10, 0);
            System.out.println(division);
        } catch(ArithmeticException e){
            System.out.println("Excepcion aritmetica arrojada: " );
            e.printStackTrace();
        }

    }
}
```

Ejemplo:

Cual es la salida por pantalla que produciría el siguiente programa:

```
public class EjemploExcepciones {
   public static int devuelveNumero(int num) {
      try {
         if (num % 2 == 0) {
            throw new Exception("Lanzando excepcion");
         }
         return 1;
      } catch (Exception ex) {
         return 2;
      } finally {
         return 3;
      }
   }
   public static void main(String[] args) {
      System.out.println(devuelveNumero(1));
   }
}
```

Solución:

Se mostrará por pantalla el valor 3, ya que es el que devuelve el bloque finally{}. Este bloque es opcional, pero si se incluye, sus sentencias se ejecutan siempre. (También mostraría el valor 3 si la llamada al método devuelveNumero se realiza con un valo par)

Actividad

Ejercicio 1

Escribe un programa que lance y capture una excepción.

El programa abre un bucle try{} en el que comienza mostrando un mensaje por pantalla. A continuación, crea un objeto de la clase Exception, indicando en su constructor un mensaje explicativo. Se procede a continuación a su lanzamiento utilizando la sentencia throw. El bloque catch{} asociado al bloque try{} anterior constituye el manejador de la excepción e incluye las sentencias necesarias para su tratamiento. En este caso, se limita a mostrar por pantalla el mensaje contenido en el objeto excepción capturado con ayuda del método getMessage().

Ejercicio 2

Escribe un programa que juegue con el usuario a adivinar un número. La computadora debe generar un número entre 1 y 500, y el usuario tiene que intentar adivinarlo. Para ello, cada vez que el usuario introduce un valor, la computadora debe decirle al usuario si el número que tiene que adivinar es mayor o menor que el que ha introducido el usuario. Cuando consiga adivinarlo, debe indicárselo e imprimir en pantalla el número de veces que el usuario ha intentado adivinar el número. Si el usuario introduce algo que no es un número, debe indicarlo en pantalla, y contarlo como un intento.

Para jugar, se guarda en una variable un número aleatorio entre 1 y 500. Se solicita al usuario un número mientras no haya acertado. Como al menos hay que pedirlo una vez, el bucle más apropiado es un bucle do-while. También hace falta un segundo bucle do-while dentro del anterior para controlar que la lectura que se realiza es realmente un número entero, y en caso contrario, dar un mensaje de error y volver a pedirlo. Scanner indica que no ha conseguido reconocer la entrada lanzando la excepción InputMismatchException. Para saber si cada vez que se pide un número se ha conseguido leer un número entero, se utiliza una variable boolean que indique si se ha leído correctamente o no.

INSTRUCCIONES: Lee la siguiente fábula y al finalizar, elabora una síntesis, un dibujo, una escultura, un video, etc. Toma una foto mientras lees para que la envíes junto con tu producto como evidencia al docente.

LA METAMORFOSIS DE PIKTOR

Apenas había caminado unos pasos por el paraíso cuando Píktor se dio de bruces con un árbol que era hombre y mujer a la vez. Saludo al árbol con deferencia y dijo:

- ¿Eres tú el árbol de la vida?

Pero cuando vio que quien se aprestaba a responder era la serpiente en lugar del árbol, dio media vuelta y prosiguió su camino. Era todos ojos: ¡Le gustaba todo tanto! Sintió intensamente que se encontraba en la fuente y origen de la vida.

Se topó con otro árbol, que era sol y luna a la vez. Y dijo Píktor:

- ¿Eres tú el árbol de la vida?

El sol asintió riendo, la luna asintió sonriendo.

Las flores más maravillosas os miraban, con los colores y reflejos más variados, con los ojos y los rostros más diversos. Algunas asentían riendo, otras asentían sonriendo, otras ni sonreían: callaban arrobadas, ensimismadas, como en su propio aroma ahogadas. Una cantaba la canción de las lilas, otra la canción de cuna azul marino. Una flor tenía unos inmensos ojos azules, otra le recordó su primer amor. Otra olía al jardín de la infancia, su perfume suave resonaba como la voz de su madre. Otra se burló de él y le sacó la lengua, una lengua muy roja y arqueada. La lamió, tenía un sabor fuerte y silvestre, sabía a resina y a miel, y también a beso de mujer.

Allí estaba Píktor, entre todas las flores desbordantes de nostalgia y de temerosa alegría. Su corazón apesadumbrado latía con fuerza, como si fuera una campana; ardía en deseo por lo desconocido, presintiendo un encantamiento.

Píktor vio un pájaro sentado, lo vio en la hierba posado, y de mil colores pintado; de todos los colores parecía el hermoso pájaro estar dotado. Preguntó al hermoso pájaro multicolor:

-Dime, ¡oh, pájaro! ¿Dónde está la felicidad?

-La felicidad- dijo el hermoso pájaro riendo con su pico de oro-, la felicidad, amigo mío, no hay donde no se halle, en la montaña y en el valle, y se encuentra por igual en la flor y en el cristal.

Tras estas palabras, el pájaro risueño sacudió su plumaje, estiró el cuello, meneó la cola, guiñó el ojo, volvió a reír, y después permaneció inmóvil, sentado en la hierba y, mira por dónde, el pájaro quedó convertido en una flor multicolor, sus plumas transformadas en hojas y sus patas en raíces. Con sus resplandores, y el fulgor de sus colores, era ahora flor entre las flores. Píktor se lo quedó mirando maravillado.

Y justo después, el pájaro-flor sacudió sus hojas y sus hilos de polvo, ya estaba harto del reino de las flores. Dejó de tener raíces, se movió con suavidad, y lentamente se elevó por los aires; se había convertido en una mariposa que se balanceó sin peso ni luz, como un ente reluciente. Píktor se quedó maravillado.

Pero la nueva mariposa, el risueño pájaro-flor-mariposa multicolor de rostros resplandeciente, revoloteó en torno al asombrado Píktor, relampagueó como el sol, y después se dejó caer suavemente caer como un copo ingrávido

a tierra, pegadito a los pies de Píktor, respiró tiernamente, se estremeció ligeramente agitando sus alas deslumbrantes, y en el acto se transformó en un cristal de colores cuyas aristas desprendían una luz rojiza. Sobre la hierba verde, la gema rojiza resplandecía maravillosamente con la claridad de un alegre repique de campanas. Pero parecía como si su hogar, las entrañas de la tierra, la estuviera llamando, pues muy pronto se volvió diminuta, a punto de desaparecer.

Entonces Píktor, presa de un deseo irresistible, se apoderó de la piedra minúscula. Maravillado contemplaba su mágico resplandor que parecía un anticipo de todas las dichas que iban a colmar su corazón.

De repente, la serpiente se enrosco en la rama de un árbol muerto y le susurró al oído:

-Esta piedra te metamorfoseará en lo que tú quieras. Dile rápido tu deseo, ¡antes de que sea tarde!

Píktor se sobresaltó y tuvo miedo de que se le escapara su felicidad. Rápidamente pronunció la palabra y se metamorfoseó en árbol. Pues ya había soñado alguna vez con ser árbol, porque los árboles le parecían la encarnación de la placidez y de la fuerza, de la dignidad.

Píktor se convirtió en árbol. Sus raíces se hundieron en la tierra y creció en altura, y de sus miembros brotaron ramas y hojas. Estaba la mar de satisfecho con su suerte. Sus fibras sedientas absorbieron el frescor profundo de la tierra y sus hojas ligeras se mecieron allá arriba en el azul del cielo. Los insectos instalaron su morada en su corteza, a sus pies anidaron liebres y erizos y pájaros en sus ramas.

El árbol Píktor era feliz y no contaba los años que iban transcurriendo. Pasaron muchos años antes de que se diera cuenta de que su felicidad no era perfecta. Poco a poco, sólo lentamente, fue aprendiendo a considerar las cosas con los ojos de un árbol. Por fin, acabó viéndolo todo claro y se puso triste.

Vio que casi todos los seres a su alrededor, en el paraíso, se metamorfoseaban con frecuencia, e incluso que todo discurría con una corriente mágica de eterna metamorfosis. Vio flores que se transformaban en piedras preciosas, o que alzaban el vuelo convertidas en resplandecientes pájaros. Vio muy cerca de él a muchos árboles que de repente desaparecían: uno se había fundido en un manantial, otro se había transformado en cocodrilo, otro, convertido en pez, nadaba alegre y feliz, desbordante de voluptuosos deseos, y pletórico se lanzaba a nuevos juegos con renovadas energías. Había elefantes que intercambiaban su ropaje con rocas, y jirafas su cuerpo con flores.

Pero él, el árbol Píktor, permanecía inalterable, él no podía ya metamorfosearse. Desde que había tomado conciencia de su inmutabilidad, toda su felicidad se había volatilizado; empezó a envejecer, y cada vez fue adoptando esa actitud cansada, seria y preocupada que suele observarse en la mayoría de los árboles viejos. También suele observarse en los caballos, los pájaros, los humanos y en todas las criaturas: cuando no poseen el don de metamorfosearse, se sumen en el tiempo con tristeza y en la preocupación y acaban perdiendo su belleza y hermosura.

Pero un día pasó por aquel rincón del paraíso una joven de rubios cabellos vestida de azul. Entre canciones y bailes, la hermosa rubia corría entre los árboles, y hasta entonces jamás se le había ocurrido plantearse si deseaba poseer el don de la metamorfosis.

Más de un monosabio sonreía a sus espaldas, algunos matorrales la acariciaban con sus ramas, algún que otro árbol le tiraba una flor, o una nuez, o una manzana sin que ella le hiciera el más mínimo caso.

Cuando el árbol Píktor vio a la joven, una nostalgia inmensa se apoderó de él, un ansia de felicidad como no la había conocido hasta entonces. Y al mismo tiempo se sumió en una profunda reflexión, pues le pareció oír su propia sangre que le gritaba:

-¡Acuérdate! Acuérdate de toda tu existencia en este momento. Encuéntrale el sentido, si no será demasiado tarde y nunca jamás volverás a encontrar la felicidad.

Y obedeció. Lo recordó todo, su origen, sus años de ser humano, su mudanza al paraíso y muy particularmente aquel instante en el que se había metamorfoseado en árbol, aquel instante maravilloso en el que había tenido la piedra mágica en la palma de la mano. En aquel momento, cuando todas las posibilidades de metamorfosis se abrían ante él, ¡nunca antes había ardido así en su interior la vida! Pensó en el pájaro que se había reído, en el árbol que era sol y luna a la vez.

Tuvo entonces la intuición de que antaño algo se le había escapado, de que había olvidado algo y de que la serpiente no le había aconsejado bien.

La muchacha oyó un murmullo en las hojas del árbol Píktor. Alzó la mirada y la embargaron, con un repentino dolor de corazón, nuevos pensamientos, nuevas ansias, nuevos sueños que despertaban dentro de su ser. Impulsada por una fuerza desconocida, se sentó al pie del árbol. Le pareció muy solitario, solitario y triste, no obstante, hermoso, conmovedor y noble en su silenciosa tristeza. Seductora le sonó la suave melodía del murmullo tembloroso de su copa. Apoyó su cuerpo contra el tronco rugoso, sintió que el árbol se estremecía profundamente, sintió el mismo estremecimiento en su propio corazón. Un extraño dolor percibió en su corazón; corrían las nubes por el cielo de su alma; y lentamente unas lágrimas pesadas fluyeron de sus ojos. ¿Qué estaba pasando? ¿Por qué tanto sufrimiento? ¿Por qué anhelaba su corazón salírsele del pecho para saltar hacia él y fundirse en él, en el hermoso árbol solitario?

El árbol se estremeció suavemente hasta la raíz, debido al esfuerzo realizado para concentrar toda su fuerza vital y proyectarla hacia la muchacha, en el abrazador anhelo de la unión. ¡Ay! ¡Haberse dejado engañar por la serpiente y haberse convertido para siempre en un árbol solitario!¡ Qué ciego, qué insensato había sido! ¿Acaso tan ignorante había sido, tan ajeno al secreto de la vida había permanecido? No, ya lo había intuido oscuramente entonces, confusamente ya lo había presentido - ¡Ay, con qué pesar recordó y comprendió entonces al árbol que era hombre y mujer a la vez!

Pasó volando un pájaro, era rojo y verde el pájaro que pasó, y alrededor del árbol voló, el hermoso y valiente pájaro. La muchacha lo siguió con la mirada, vio que de su pico caía algo, rojo como la sangre, rojo como las brasas, que caía y relucía en la hierba verde, con unos destellos rojos tan poderosos que la muchacha se agachó, y en la hierba la piedra roja recogió. Era un carbunclo, era un rubí, y donde hay un carbunclo, oscuridad no puede haber allí.

Apenas la muchacha hubo recogido la piedra mágica en su mano blanca que el deseo anhelado que henchía su corazón se realizó. La joven se volatilizó, se fundió, formó una sola cosa con el árbol. Una rama joven y vigorosa brotó del tronco y deprisa se disparó hacia arriba hasta él.

Ahora todo estaba como ha de estar, todo estaba en su lugar, el mundo estaba en orden, por fin había encontrado el paraíso. Píktor dejó de ser árbol viejo y preocupado. Ahora cantaba a voz en grito: ¡Piktoria! ¡Victoria!

Estaba metamorfoseado. Y debido a que, esta vez, por fin había sabido encontrar la metamorfosis eterna, debido a que una mitad se había hecho un todo, a partir de aquel momento podía seguir metamorfoseándose cuanto quisiera. La corriente mágica del devenir fluyó perenne por sus venas y para siempre formó parte de la constante y permanente creación eterna.

Se transformó en ciervo, se transformó en pez, se transformó en ser humano y en serpiente y también en nube y en pájaro. Pero bajo cualquier apariencia, siempre formó un todo, una pareja, sol y luna, hombre y mujer, y como ríos gemelos fluyó a través de las tierras y como estrellas gemelas brilló en el firmamento.

Conclusión

El trabajo en casa, permite, reforzar y recapitular, conocimientos, ideas, prácticas y ejercicios que le otorgan al alumno destrezas y habilidades para comprender y desempeñar, hoy y a futuro, su carrera como profesional técnico bachiller en informática, dentro del subsistema CONALEP.

Academia de Informática
Conalep - Xalapa

Nota aclaratoria: el contenido de este material es una guía de apoyo para el alumno. Si se presentan dudas el estudiante podrá dirigirse, mediante un medio de comunicación, al docente para aclarar sus dudas, si es que se presentan.

Yo@prendo informáticos UNIDOS;
{PROYECTO ACADÉMICO}
• modalidad **cuadernillos** •