UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

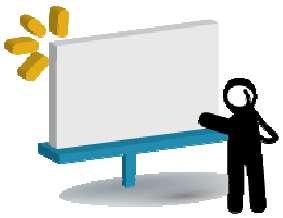
Centro de e-Learning

FUNDAMENTOS DE LA PROGRAMACIÓN

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

MÓDULO 3 - UNIDAD 9

Programación Orientada a Objetos



Presentación:

Con esta Unidad 9 comenzamos con el Módulo 3 del curso, que va a estar enfocada en el paradigma de Programación Orientada a Objetos (POO).

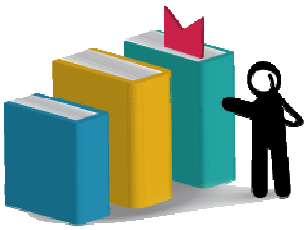
Comenzamos analizando los principales componentes del paradigma, mostrando las principales diferencias con el paradigma Estructurado, aunque también reutilizando muchos de los conceptos vistos anteriormente.

Posteriormente comenzaremos a analizar los principios que rigen el paradigma, tarea que finalizaremos en la próxima Unidad.

Objetivos:

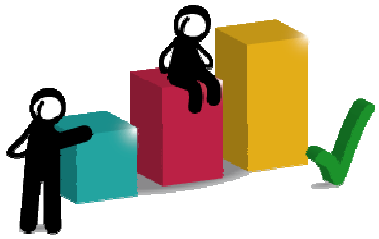
Que los participantes:

- Obtengan los primeros conceptos del paradigma de Programación Orientada a Objetos (POO)
- Incorporen los conceptos de clase, objeto, métodos y atributos
- Incorporen los conceptos de los dos primeros principios que hacen a la Orientación a Objetos (OO): abstracción y encapsulamiento



Bloques temáticos:

1. El paradigma Orientado a Objetos
2. Clases
3. Objetos
4. Métodos y atributos
5. Principios de la OO: Abstracción
6. Principios de la OO: Encapsulamiento



Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

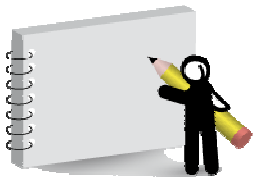
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

* El MEC es el modelo de E-learning colaborativo de nuestro Centro.



Tomen nota

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

1. El paradigma Orientado a Objetos

Como lo indica el hecho de ser un paradigma, la orientación a objetos (o programación orientada a objetos, de ahora en más indistintamente POO u OO), está relacionado con cómo hacer las cosas por sobre cómo funcionan las cosas. Dicho de otra manera, está relacionado a ciertas prácticas con sustento teórico por sobre las herramientas en las cuales se implementan estas prácticas.

Veremos a lo largo de este Módulo cómo se reutilizarán muchos de los conceptos del Módulo anterior, ya que, tanto en la programación estructurada como en la POO, las estructuras condicionales, las iterativas, el uso de la indentación, entre otras, siguen estando presentes. También veremos que existen similitudes conceptuales con respecto a otros temas vistos, como las funciones, aunque con nombres, usos y conceptos de base distintos.

Como dijimos antes, la OO (como concepto) no tiene que ver con las herramientas en sí, aunque nos permitiremos algunas licencias relacionadas a implementaciones puntuales con el fin de presentar ejemplos similares a los del día a día laboral de un profesional de la programación. Estas licencias estarán orientadas a permitir una transición simple y natural a las implementaciones de los lenguajes de POO más difundidos en el mercado (Java, C/C++/C#, entre otros) reduciendo la brecha de conocimiento y la curva de aprendizaje, a la vez que se mantienen sólidos fundamentos teóricos.

Dentro de la OO, los Objetos en sí son representaciones de distintos aspectos de la realidad. Como venimos diciendo, uno de los objetivos de la programación en general es modelar una realidad con el fin de resolver un problema. En este caso, los objetos cumplen este concepto a la perfección, por sus características y su concepción.

Por lo tanto, podemos decir que los objetos son las entidades que modelan esa realidad y tienen un estado, comportamiento e identidad particulares.

- El **estado** de un objeto se compone de los datos que puede contener a través de atributos o propiedades.

- El **comportamiento** de un objeto está dado por todas aquellas acciones que un objeto puede realizar, y que se encuentran definidas por los métodos o mensajes que puede enviar. Dicho de otra forma, son las operaciones (en general, no solamente aritméticas) que puede realizar un objeto.

Los conceptos de atributo/propiedad y método/mensaje los veremos más adelante en esta misma Unidad.

- La identidad de un objeto es un identificador que lo hace único. Este concepto será recursivamente retomado en todo el Módulo.

La complejidad de la OO pasa por un "forma de pensar" los problemas, término que repetíamos al comienzo de curso, y que está relacionada con la subjetividad de quien programa: la manera en la que hace las cosas y pueden ser diferentes entre distintos programadores, aunque siempre debe estar regida (esta forma de pensar) por el sentido común, la simplicidad y ciertas las buenas prácticas. Programar orientado a objetos no es lo difícil. Difícil es programar bien. Y programar bien es importante ya que de esta forma podemos explotar mejor todas las ventajas de la POO.

Pensar en términos de objetos es muy parecido a cómo concebimos y percibimos lo que nos rodea en la vida real. Vamos a pensar (sin intentar ser demasiado originales en el ejemplo) en un auto, para tratar de modelarlo en Objetos. En forma narrativa, **podemos decir que "auto" es el elemento principal de interés, que engloba toda nuestra problemática, y que a la vez tiene una serie de particularidades: color, modelo o marca, entre otros. Además, puede realizar una serie de acciones: ponerse en marcha, frenar, estacionar, etc.**

En POO el **auto** sería el objeto, los **atributos** serían las características como el color o el modelo y los **métodos** serían las funcionalidades asociadas como ponerse en marcha o frenar, temas que veremos en detalle a continuación.

Otro ejemplo podría tratar de modelar en OO una **fracción**, es decir, esa estructura matemática que tiene un numerador y un denominador que divide al numerador, por ejemplo 1/2. En este caso, la **fracción** es el objeto y tiene dos **atributos**, el numerador y el denominador. Posteriormente podría definir varios **métodos** que modifican su comportamiento, como simplificarse, sumarse con otra fracción, restarse con otra fracción, o cualquier otro que tenga sentido para una fracción.

Estos objetos se podrán utilizar en los sistemas, por ejemplo, en un programa de matemáticas se podría hacer uso del objeto fracción y en un programa que gestione un concesionario se podrá utilizar el objeto auto. Los sistemas OO utilizan muchos objetos para realizar las acciones que se desea realizar y ellos mismos también son objetos. Es decir, el concesionario de autos será un objeto que utilizará objetos autos, vendedores, formularios, etc.

2. Clases

Si bien venimos hablando de objetos, existe un artefacto "anterior" (viendo la POO como un secuencia de conocimientos) que es necesario conocer: **la clase**.

Comenzaremos con una famosa y muy difundida definición que nos permitirá comprender las diferencias entre clase y objeto:

“Un objeto es una instancia de una clase”

Esta definición, nos dice que **primero existe la clase y que a partir de ella se crean los objetos**. Aquí podríamos hacer un paralelismo diciendo que una clase es un molde y el objeto es el producto moldeado.

En OO, tiende a quedar en desuso el concepto de “programa” (como ente que agrupa código fuente, algoritmos, etc.), siendo reemplazado por el concepto de clase.

A continuación veremos un ejemplo de una clase, continuando con el ejemplo del auto.

```
clase Auto
    // declaración de atributos de la clase
    Color colorDelAuto nuevaInstancia Color()
    Marca marcaDelAuto nuevaInstancia Marca()

    //declaración de métodos
    metodo definirColor (Color color)
        // atributos del método
        // instrucciones del método

    fin metodo

    metodo definirMarca(Marca marca)
        // declaración de atributos del método
        // instrucciones del método

    fin metodo
fin clase
```

A simple vista podemos observar algunas diferencias con lo que veníamos viendo hasta ahora en Estructurado:

- Vemos que introducimos nuevas palabras reservadas:
    - “clase”, “fin clase”
    - “metodo”, “fin metodo”

- Eliminamos:
  - en la declaración de atributos (antes variables) el uso de la palabra reservada “var”.
  - además de "programa", "inicio" y "fin".
- El nombre de la clase (antes era un programa, ahora es una clase) se escribe por convención en UpperCamelCase.
- No tenemos variables de "tipo primitivo" (integer, string, etc.). Si bien la mayoría de los lenguajes de POO implementan los tipos primitivos, de momento haremos honor a que “en objetos, todo es objetos”.

Una variable de "tipo primitivo" nos permite guardar un dato, por ejemplo: *var string lugar = "Madrid"*

En la variable "lugar" solo podemos guardar un único dato. En un arreglo, ocurre algo similar: solo podemos guardar datos. Pero en POO, en cada Objeto no solo vamos a poder guardar datos (en los atributos) sino tener comportamiento, acciones (en los métodos). Y, además, ¡vamos a poder crear nuestros propios Objetos para lo que necesitemos!

- Aquí comenzaremos con algunas de las licencias que dijimos que nos tomaríamos: vamos a dar por sentado (como ocurre en la realidad) que algunas cuestiones básicas vienen dadas por los lenguajes. Por ejemplo: si queremos asignar una cadena de caracteres, no vamos a preocuparnos de tener que bajar a nivel de lenguaje de máquina para pedir una composición de bit y bytes, etc. Simplemente daremos por hecho que los lenguajes (en nuestro caso, un pseudocódigo) ya nos proveen alguno de estos elementos básicos. Por lo que mantendremos nuestros tipos de datos, sólo que los usaremos en para nombrarlos el UpperCamelCase para indicar que son clases. Por eso, tendremos disponibles estas clases: String, Integer, Boolean, Date y Float. Que “modelan” cadenas de caracteres, números enteros, valores binarios: verdadero y falso, fechas y números con decimales.
- Seguimos manteniendo 2 secciones:

- En la superior o inicial se definirán todos los atributos (antes eran "variables") de la clase
- Posteriormente se definirán los métodos (antes eran "funciones") de la clase (antes era "programa")

- Finalmente, vemos que los atributos fueron declarados como clases. Vemos que son 2:
  - Color
  - Marca

Por lo que tenemos estas otras dos nuevas clases que deberíamos escribir, para que el ejemplo esté completo.

```
clase Color
    String color nuevaInstancia String()

    metodo definirColor(String colorNuevo)
        color = colorNuevo
    fin metodo

    metodo string devolverColor()
        retornar: color
    fin metodo

fin clase
```

```
clase Marca
    String marca nuevaInstancia String()

    metodo definirMarca(String marcaNueva)
        marca = marcaNueva
    fin metodo

    metodo String devolverMarca()
        retornar: marca
    fin metodo
fin clase
```

Ambos casos son similares. Podemos ver que lo que antes se trataba de tipo de dato primitivo ("color" y "marca", ahora se convierte en un objeto. Ambos objetos tienen un atributo que define su característica y dos métodos: el primero utilizado para cambiar el color y el segundo para devolver el color actual del auto. Más adelante, en esta misma Unidad, veremos más detalles sobre el uso de atributos y métodos.

También vemos que los atributos (antes variables) se definen de una manera particular, no tan simple a cómo se creaban variables en Estructurado:

```
String marca nuevaInstancia String()
```

Donde tenemos:

- "String" es el tipo de dato del atributo. Es la CLASE a partir de la cual se crea el objeto
- "marca" es EL objeto

- "nuevaInstancia" se utiliza para aclarar que se está creando un objeto
- "String()" es el Constructor (ya veremos de qué se trata esto) usado para crear el objeto

Sobre los métodos:

```
metodo definirMarca(String marcaNueva)
    marca = marcaNueva
fin metodo
```

Se declara un método llamado "definirMarca" que recibe como parámetro un tipo "String". Dentro del método se asigna el valor del parámetro "marcaNueva" al atributo de clase "marca". Importante: deben coincidir los tipos (la clase del parámetro y la clase con se declaró el atributo) para que no haya conflictos de tipos de datos (lo mismo que ocurría en Estructurado).

```
metodo String devolverMarca()
    retornar: marca
fin metodo
```

En este caso vemos un método llamado "devolverMarca" que no recibe parámetros, pero sí devuelve un valor del tipo String. En este caso, devuelve el valor del atributo "marca".

Más adelante, en esta Unidad, veremos más en profundidad todos estos conceptos, incluida la creación de atributos.

3. Métodos y Atributos

Como ya dijimos, un método o "mensaje" es la analogía en OO a las funciones del paradigma estructurado. Su objetivo es darle comportamiento a un objeto a través de la ejecución de instrucciones de negocio, en el sentido de “hacer algo” para lo cual fue concebida la clase que viene a resolver un problema en particular, de un “negocio” (industria, sector, conocimiento, etc.) en particular. Además, modifican los atributos de un objeto, cambiando su estado.

Los métodos devuelven un único valor (o ninguno) y reciben una cantidad ilimitada de parámetros, dependiendo de la implementación del lenguaje en particular.

Se suele decir que un método es “invocado” o “llamado”, indistintamente cuando es utilizado.

También existen métodos especiales llamados constructores, que en lenguajes como Java, .NET o C++ reciben el mismo nombre que la clase, aunque estos no están presentes en todos los lenguajes OO. Más adelante, en esta unidad ampliaremos este concepto.

Los atributos o propiedades de un objeto conforman el estado del mismo. Estos, a su vez, deberían ser siempre objetos, desde el punto de vista más purista de la POO, aunque existen lenguajes híbridos (como Java) que permiten definir atributos de tipos primitivos.

Otra de las premisas, es que los atributos nunca deberían ser accedidos en forma directa, aunque sea posible hacerlo. La forma de accederlos o modificarlos debería ser a través de métodos. Estos métodos especiales que reciben el nombre de getters y setters, en ingles, normalmente traducidos como “obtener” (del “get”, en inglés) y “asignar”, (del “set”, en inglés) aunque sin difusión en su forma traducida.

Como regla general, los métodos setters y getters tienen las siguientes características:

	Getter	Setter
Recibe parámetros	No	Sólo uno, para modificar un único atributo
Devuelve valores	Sí, del atributo al que corresponde	No

Un ejemplo de implementación de estos getters y setters es el siguiente:

```
clase Compania
    //atributos
    String nombre nuevaInstancia String()
    String domicilio nuevaInstancia String()

    //contructorp or default
    metodo Compania()
    fin metodo

    //getters
    metodo String obtenerDomicilio()
        retornar: domicilio
    fin metodo
    metodo String obtenerNombre()
        retornar: nombre
    fin metodo

    //setters
    metodo asignarDomicilio(String domicilioNuevo)
        domicilio = domicilioNuevo
    fin metodo
    metodo asignarNombre(String nombreNuevo)
        nombre = nombreNuevo
    fin metodo
fin clase
```

Tanto los atributos como los métodos se acceden o invocan a través del operador punto (".")

Siguiendo con el último ejemplo:

```
Clase Club
    String nombre nuevaInstancia String()

    metodo Club()
        nombre = "CARP"
    fin metodo

    metodo String obtenerNombre()
        retornar: nombre
    fin metodo

fin clase

Clase UsaClub
    metodo UsaClub()
        //creo el objeto
        Club miClub nuevaInstancia Club()

        //accedo al atributo "nombre" del objeto
        mostrar: miClub.nombre
        //se muestra "CARP"

        //invoco al método "obtenerNombnre"
        mostrar: miClub.obtenerNombre()
        //se muestra "CARP"

    fin metodo
fin clase
```

- Las expresiones "miClub.nombre" y " miClub.obtenerNombre()" son equivalentes: se ambos se obtiene el valor del atributo "nombre".
    - Si bien ambas instrucciones tienen la misma función, la forma correcta de obtener el valor de un atributo sería a través del método.

- Vemos que ambos casos se utiliza el operador punto (".") para conectar el objeto ("miClub") con el atributo ("nombre") y el método ("obtenerNombre()")

4. Objetos

Dijimos que un objeto es una instancia de una clase. ¿Qué significa esto? Que la clase en sí no tiene comportamiento ni estado, sino que lo tiene el objeto. Y que, a partir de la clase, con una instrucción en particular, creamos **objetos, que son los que tienen estado y comportamiento**. Una clase es un conjunto de instrucciones. Aunque también es cierto que para crear una instancia de una clase necesitamos instrucciones. Y un mecanismo para realizar esa creación.

A continuación veremos un ejemplo de creación de instancias (objetos) a partir de las clases que definimos arriba.

```
clase Auto
    Color colorDelAuto nuevaInstancia Color()
    Marca marcaDelAuto nuevaInstancia Marca()
    Float costo nuevaInstancia Float()
    Float precio nuevaInstancia Float()

    //constructor
    metodo Auto ()
        costo = 100000,00
    fin metodo

    //métodos de negocio
    metodo definirColor (Color color)
        colorDelAuto = color
    fin metodo

    metodo definirMarca(Marca marca)
        marcaDelAuto = marca
    fin metodo

    metodo calcularPrecio()
        precio = costo * 1,5
    fin metodo

    //otros métodos...
fin clase
```

Supongamos que tengo una clase principal, de ejemplo, que requiere crear dos autos y asignarles distintos colores y marcas a cada uno. Para eso definiremos la clase "EjemploAuto", pero primero ampliaremos el desarrollo de la clase Auto, definiendo el comportamiento, completando la estructura básica que habíamos visto antes.

Aquí completamos la implementación de la clase Auto, definiendo el comportamiento a través de los métodos antes definidos.

Ahora, analizaremos la clase de ejemplo, que utiliza las 3 clases vistas ahora (la nueva de Autos y las dos anteriores de Color y Marca, que no cambiaron).

```
clase EjemploAuto
    metodo EjemploAuto()
        //primer objeto de tipo Auto creado
        Auto primerAuto nuevaInstancia Auto()

        Color primerAutoColor nuevaInstancia Color()
        Color auxColor nuevaInstancia Color()
        Marca primerAutoMarca nuevaInstancia Marca()
        Marca auxMarca nuevaInstancia Marca()

        primerAutoColor.definirColor("gris plata")
        auxColor = primerAutoColor
        primerAutoMarca.definirMarca("VW")
        auxMarca = primerAutoMarca

        primerAuto.definirColor(auxColor)
        primerAuto.definirMarca(auxMarca)

        //segundo objeto de tipo Auto creado
        Auto segundoAuto nuevaInstancia Auto()
        Color segundoAutoColor nuevaInstancia Color()
        Marca segundoAutoMarca nuevaInstancia Marca()

        segundoAutoColor.definirColor("scandium")
        segundoAuto.definirColor(segundoAutoColor)

        segundoAutoMarca.definirMarca("Fiat")
        segundoAuto.definirMarca(segundoAutoMarca)

    fin metodo
fin clase
```

Vamos por partes:

- "metodo EjemploAuto()"
    - **Cuando un método tiene el mismo nombre que la clase (incluidos mayúsculas y minúsculas) pasa a llamarse "constructor".** Esos métodos (puede haber más de uno por clase) **nunca devuelven valores y son los primeros que se ejecutan (y se ejecutan siempre) cuando una clase es instanciada**, o sea, cuando se crea un objeto. **Este método se suele utilizar para inicializar el objeto**: asignarle características "por default" o realizar alguna acción en particular. En nuestro caso, queremos que cada vez que se instancia a la clase "EjemploAuto" se ejecute esa porción de código. En el caso de la clase "Auto" vemos cómo se usa su constructor para darle un valor inicial al atributo "costo": esto sucede cada vez que se instancia (crea un objeto) a partir de la clase "Auto".
- "Auto primerAuto nuevaInstancia Auto()"
    - En esta instrucción se crea un objeto a partir de la clase Auto, y se hace utilizando el constructor defecto ("Auto()"). Lo mismo ocurre cuando se instancian "primerAutoColor", "segundoAutoMarca", etc.
- "Marca auxMarca nuevaInstancia Marca()"
    - Este objeto del tipo Marca y su similar de tipo Color se crean como objetos temporales.
- "primerAutoMarca.definirMarca("VW")"
    - En este caso, le damos un valor a "primerAutoMarca" a través de la invocación del método "definirMarca(String)". A ese método le enviamos un valor tipo String fijo "VW"
- "auxMarca = primerAutoMarca"
    - Luego, asignamos el objeto del tipo Marca llamado "primerAutoMarca" a un objeto auxiliar llamado "auxMarca".
- "primerAuto.definirMarca(auxMarca)"
    - Caso similar al anterior, estamos enviando un objeto de tipo Marca llamado auxMarca creado en el punto anterior, con un valor determinado, a la

primera instancia de la clase Auto, llamada “primerAuto” a través de uno de sus métodos, llamado “definirMarca”. Este método (ver clase Auto) recibe un único parámetro de tipo Marca, que es lo que le estamos enviando y no devuelve valores (ésta es la diferencia esencial con la instrucción anterior).

- En el caso primerAutoColor se realizar una operatoria similar.
- En la segunda parte del ejercicio, se realizar una operación similar aunque se evita el uso de objetos auxiliares.

En este ejemplo vimos cómo se crearon dos instancias de la clase Auto y cómo a cada una se le asignaron valores distintos de color y marca, utilizando las clases Color y Marca, respectivamente.

A continuación podemos ver un ejemplo de uso de diversos constructores:

Repasamos la definición: un constructor es un método especial, que *se llama exactamente igual que la clase* que lo contiene, no devuelve valores, se ejecuta siempre que se crea un objeto y se utiliza cuando se instancia una clase.

```
clase Auto
    Color colorDelAuto nuevaInstancia Color()
    Marca marcaDelAuto nuevaInstancia Marca()

    //CONSTRUCTOR por defecto
    //****************************************** */
    metodo Auto()
        // en este caso no se hace nada
    fin metodo
    //****************************************** */

    //CONSTRUCTOR parametrizado
    //****************************************** */
    metodo Auto(Color color, Marca marca)
        // acciones de inicialización del objeto
        definirColor(color)
        definirMarca(marca)
    fin metodo
    //****************************************** */

    //métodos de negocio
    metodo definirColor (Color color)
        colorDelAuto = color
    fin metodo

    metodo definirMarca(Marca marca)
        marcaDelAuto = marca
    fin metodo

    metodo calcularPrecio()
        precio = costo * 1,5
    fin metodo

fin clase
```

Cuando se quisiera instanciar esta clase, se haría a través del uso de la sentencia

"… nuevaInstancia Auto(color, marca)"

Siendo "color" y "marca" instancias de esas clases creadas anteriormente.

```
Clase CreoAutos
    metodo CreoAutos()
        //uso el constructor por default de la clase Auto
        Auto autoPorDefault nuevaInstancia Auto()
        autoPorDefault.calcularPrecio()

        //creo instancias que van a ser los parámetros
        Color miColor nuevaInstancia Color()
        miColor.definirColor("blanco perlado")
        Marca miMarca nuevaInstancia Marca()
        miMarca.definirMarca("Nissan")

        //uso el constructor parametrizado de la clase Auto
        Auto autoParametrizado nuevaInstancia Auto(miColor, miMarca)
        autoParametrizado.calcularPrecio()
    fin metodo
fin clase
```

En ambos casos creo instancia de la clase "Auto", o sea, 2 objetos:

- autoPorDefault
- autoParametrizado

Luego, invoco al método "calcularPrecio" de cada objeto.

La diferencia entre ambos objetos será que el primer auto ("autoPorDefault") no tiene características propias, lo que sí ocurre con el segundo auto ("autoParametrizado") que tiene el color y la marca establecidos.

¿Porqué la clase "Auto" tiene dos atributos de tipo "Marca" y "Color" y no son de tipo "String", por ejemplo?

El modelo de Programación Orientada a Objetos nos hace pensar en ENTIDADES. El programador debe identificar cuáles son las entidades importantes y cómo deben ser modeladas.

En el ejemplo del Auto, se considera que Marca y Color son elementos que se puede llegan a destacar y pueden requerir, por ejemplo, tener su propia lógica de negocio: un comportamiento propio, estado propio: métodos y atributos, no solamente un dato.

Pensemos en el dato "CUIL" (Código Único de Identificación Laboral, en Argentina). Es un código formado por 2 cifras numéricas, luego un número de 10 cifras (normalmente) y finalmente otros número de 2 cifras, separados estos tres con guiones. Entonces, se podría pensar como un dato tipo String (por la presencia de los caracteres alfanuméricos "-"), por ejemplo:

```
String cuil nuevaInstancia String()
cuil = "20-25987584-8"
```

Como un dato para identificar unívocamente a un cliente, es suficiente y está bien.

Pero ¿qué pasa si somos la ANSES (*)?

Sabemos que el CUIL tiene un dígito verificador, por lo tanto, ¿podemos validar si un CUIL es válido o no, verdad?

Siguiendo la lógica anterior, podríamos hacer:

```
String cuil nuevaInstancia String()
cuil = "20-25987584-8"

Validador valida nuevaInstancia Validador()
valida.hacerValidacionDeCUIL(cuil)
```

Para esto, vamos a necesitar una clase "Validador" que tenga el método "hacerValidacionDeCUIL".

Es una alternativa.

¿Cuál podría ser otra? Que el CUIL sepa validarse así mismo (desde el punto de vista de objetos, es la opción más correcta).

Por lo tanto, en lugar de tener una clase "Validador" que tenga el método "hacerValidacionDeCUIL", se puede crear la clase CUIL que tenga ese método. Por ejemplo:

```
clase publica CUIL
        String nroCuil nuevaInstancia String()

        metodo publico CUIL()
        fin metodo

        metodo publico CUIL(String nuevoNro)
                nroCuil = nuevoNro
        fin metodo

        metodo publico Boolean validate()
                //logica de validacion
        fin metodo
fin clase
```

Y luego:

```
CUIL cuil nuevaInstancia CUIL( "20-25987584-8")
si cuil.validate() = verdadero entonces
      mostrar: "nro oK"
sino
      mostrar: "nro NO oK"
fin si
```

En este contexto el CUIL, como dato, toma una relevancia mayor, por lo que deja de ser un simple String a convertirse en una entidad en sí misma, con estado y con comportamiento.

Entonces, ¿cómo se distingue qué es un dato simple y qué una entidad? Bueno, esto va a depender del ámbito del problema en cuestión. De la abstracción y lo que el programador entienda que es relevante y merece ser tratado de forma distintiva y qué no.

(*) La ANSES, en Argentina, otorga jubilaciones y pensiones, asignaciones familiares a los trabajadores en actividad y jubilados, subsidios familiares, prestaciones por desempleo, financiadas por el Fondo Nacional de Empleo, e implementar diversos programas de Seguridad Social. Es la encargada, también, de gestionar los CUIL.

5. Principios de la OO: Abstracción

Veremos en esta Unidad y en las siguientes los principios más importantes que rigen el paradigma OO. El primero que explicamos, la abstracción, y se trata de la "habilidad" de aislar un componente de su contexto o de otros componentes relacionados.

Podemos decir que se trata de un proceso subjetivo por el cual un programador modela un objeto destacando sus datos más relevantes, dejando de lado aquellos que no aportan valor al problema que se busca solucionar. Obviamente va a depender de la visión, información disponible y parcial con la cual se depura un objeto, moldeándolo para el fin que se cree que va a ser utilizado.

Por ejemplo, si tuviéramos que construir nuestro sistema para una concesionaria, podríamos modelar el objeto Auto de las siguientes formas:

```
clase Auto
    Color color ...
    Marca marca ...
    Modelo modelo ...
    Precio precio ...
```

```
clase Auto
    Duenio duenio ...
    Modelo modelo ...
    FechaIngreso  fecha ...
    Problemas problemas ...
```

Ambas clases muestran atributos de autos. En la primera podemos ver algunos que tendrían sentido en el contexto de una concesionaria. El segundo ejemplo también tiene datos válidos de auto, aunque más similar a los que necesitarían en un taller.

No da lo mismo tener una "mega clase" con los atributos de ambos, ya que no estaríamos cumpliendo con la premisa de la abstracción.

6. Principios de la OO: Encapsulamiento

Otros de los principios que rigen la OO es el encapsulamiento. Su utilidad radica en la simplificación del manejo de la complejidad, ya que nos permite crear objetos como “cajas negras” donde sólo se conoce el comportamiento, pero no los detalles de implementación. Esto permite enfocarse en (y conocer) qué hace el objeto en de lugar de cómo lo hace, facilitando su uso.

También permite ocultar el estado, definido por los atributos, de un objeto de manera que sólo se puede cambiar mediante las operaciones explícitamente definidas para ese objeto. De esta forma se restringe el comportamiento, a la vez que se eleva el nivel de seguridad, al asegurarnos que un usuario de nuestra clase (otro programador) acceda al estado del objeto a través de los medios destinados puntualmente a esos fines.

Las formas se encapsular se pueden definir como:

- Estándar o predeterminado
- Abierto: el objeto puede ser modificado y accedido desde afuera de la clase
- Protegido: el objeto sólo puede ser accedido desde la misma clase (auto referencia) o por clases que la heredan (ampliaremos el concepto de Herencia en las próximas Unidades)
- Cerrado: el objeto sólo puede ser accedido desde el mismo objeto (auto referencia únicamente)

En la próxima unidad ampliaremos este concepto y veremos la forma concreta de aplicarlo en nuestras clases.

7. Ejemplos y tips sobre POO

- Cuando se declara un atributo, se está **creando un objeto** a partir de un *constructor* usando la nueva palabra reservada "*nuevaInstancia*" y ese objeto va a tener un *tipo de dato*. No hay que confundir el tipo de dato con el constructor, aunque ambos (de momento) lleven el mismo nombre (pero el constructor usado va acompañado de "()"). Repasando:

String nombre nuevaInstancia String()

(1) *(2)* *(3)* *(4)*

Tenemos, por orden:

  - *(1):* **String**: el "tipo de dato", la clase a partir de la cual se crea el objeto: es la clase instanciada
  - *(2):* **nombre**: el objeto creado
  - *(3):* **nuevaInstancia**: palabra reservada
  - *(4):* **String():** método constructor utilizado para crear el objeto (por eso tiene "()") que pertenece a la clase String. En este caso, es el constructor por defecto, ya que no recibe parámetros

Si en lugar de "String" tuviésemos "Color" o "Marca" o "Modelo" o "Club", sería exactamente lo mismo.

- ¿Qué podemos tener dentro de los métodos? Vimos unos tipos de métodos especiales llamados setters y getters, que guardan y devuelven datos. También vimos los métodos constructores. El resto de los métodos, los que queramos definir, se van a llamar "métodos de negocio", dado que van realizar diferentes operaciones y acciones relacionadas al problema que estemos resolviendo. Dichos métodos pueden contener todas las estructuras vimos:

- condicionales, de todos los tipos (simple, doble, múltiple, etc.)
- repetitivas (ciclos "mientras")
- acumuladores y contadores
- asignaciones y operaciones aritméticas y lógicas
- ingreso de datos en atributos ("ingresar:...")
- salida de datos ("mostrar:...") tanto de valores fijos como de valores de atributos
- vectores y matrices
- también algoritmos: podríamos implementar una búsqueda binaria o un método de ordenamiento por burbujeo usando objetos, con clases, métodos y atributos en lugar de programas, funciones y variables (¡y no cambiaría mucho!)
- Por ejemplo:

```
Clase Usuario
    String user nuevaInstancia String()
    String pass nuevaInstancia String()

    metodo Usuario()
    fin metodo

    metodo Boolean validar()
        si user = "" O pass "" entonces
            mostrar: "Error: usuario o clave inválidos"
            retornar: falso
        sino
            si user = "pobla" Y pass = "pere" entonces
                mostrar: "Acceso permitido"
                retornar: verdadero
            sino
                mostrar: "Acceso denegado"
                retornar: falso
            fin si
        fin si
    fin metodo
fin clase
```

```
Clase CrearUsuario
    Usuario u nuevaInstancia Usuario()

    metodo CrearUsuario
        cargarDatos()
        mientras u.validar() = falso hacer
            cargarDatos()
        fin mientras
    fin metodo

    metodo cargarDatos()
        mostrar: "Ingresar usuario"
        ingresar: u.user
        mostrar: "Ingresar clave"
        ingresar: u.pass
    fin metodo
fin clase
```

En la clase "Usuario" tenemos un método con una estructura condicional doble y otra anidada. En

En la clase "CrearUsuario" tenemos un método con un ciclo "mientras.

IMPORTANTE: cuando estamos invocando un método que se encuentra en la misma clase, no es necesario crear un objeto ni usar el operador punto ("."). Con colocar el nombre del método y los "()", es suficiente (no olvidar los parámetros, si es que tiene...)

Un ejemplo de esto lo podemos ver en la instrucción: "cargarDatos()"

- Tips breves:
  - No puede haber clases dentro de clases
  - No puede haber métodos dentro de métodos
  - Las clases NO van dentro de "programas"
  - Un método puede o no devolver valores, pero sólo uno como máximo si lo hace
  - Si un método declara devolver valores, si o si debe tener un "retornar:" adentro.
  - Y la inversa: si un método tiene un "retornar:" adentro si o si debe declarar en devuelve un valor. Y el tipo de dato será la clase a la que pertenece el objeto que será devuelto.
  - Cuando nos encontramos con un "retornar:" el método interrumpe su ejecución y vuelve al mismo punto desde el cual fue invocado
  - Un método puede o no recibir parámetros, y no hay límite preestablecido
  - Para invocar un método o usar un atributo que se encuentra en otra clase, si o si vamos a tener que crear un instancia de esa clase (o sea, crear un objeto) y acceder ese método o atributo usando el objeto + el operador punto (".") + el nombre de atributo o nombre del método (recordar poner los "()" y los parámetros, si es que lleva).
  - Los parámetros de los métodos SIEMPRE llevan el tipo de dato, que está dado por la clase a la que pertenece (String, Boolean, Auto, Marca, etc)
  - La palabra reservada "var" no se usa más
  - Otras palabras reservadas que vamos a dejar de usar a partir de esta unidad: "programa", "inicio", "fin", "const"

- Ejemplo:

```
//creo una clase nueva (no hay clases dentro de clases)
Clase Libro
    //defino los atributos (todos los atributos son objetos)
    String nombre nuevaInstancia String()
    String autor nuevaInstancia String()
    Integer anioPublicacion nuevaInstancia Integer()
    //el atributo está compuesto por: el "tipo de dato" (Integer), el objeto ("anioPulicacion"),
    //la palabra reservada "nuevaInstancia" y el constructor utilizada ("Integer()")

    //Integer, String, Date, Boolean, Float, etc los vamos a tomar como clases
    //existentes en nuestro pseudocódigo
    Integer anioActual nuevaInstancia Integer()

    //un método que nunca retorna valores
    //y se llama exactamente igual que la clase, es un método constructor
    //si no recibe parámetros, es el "constructor por defecto/default"
    metodo Libro()
        //uso el constructor para darle forma al objeto que voy a crear
        //en este caso inicializo los atributos con valores inválidos
        autor = "Anónimo"
        anioPublicacion = -1
        anioActual = 2020
        nombre = "S/D"
    fin metodo

    //metodo que calcula los años de antigüedad del libro
    metodo Integer calcularAntiguedad()
        retornar: anioActual - anioPublicacion
    fin metodo

    //métodos setter, para asignarle un valor a un atributo
    //el setter no devuelve valores y recibe un parámetro
    //solo hace una asignación
    metodo asignarAutor(String nuevoAutor)
        autor = nuevoAutor
    fin metodo

    metodo asignarAnioPublicacion(Integer nuevoAnioPublicacion)
        anioPublicacion = nuevoAnioPublicacion
    fin metodo

    metodo asignarNombre(String nuevoNombre)
        nombre = nuevoNombre
    fin metodo

    //metodo getter, utilizada para devolver el valor de un atributo
    //siempre retorna un valor y no recibe parámetros
    metodo String obtenerAutor()
        retornar: autor
    fin metodo
fin clase
```

- ¿Qué es una "clase de integración"?

    - También conocidas como clase "integradora" o de "test".

    - Cada vez que definamos nuestras clases (seguramente necesitemos varias para modelar una entidad compleja, como cuando vimos el caso de Auto-Color-Marca), vamos a necesitar una clase de integración (también llamada clase de "test", porque la idea es probar cómo nuestras clases se relacionan e interactúan, como vimos en los casos de las clases "CreoAutos", "EjemploAuto", "UsaClub"). En esta clase vamos a crear atributos que van a ser las instancias de nuestras clases, también realizaremos invocaciones a los métodos y usaremos los atributos. De esta forma, podremos comprobar que las partes forman un todo y que existe coherencia entre ellas.

    - También, como vemos a continuación es posible tener una clase integradora que se ocupe únicamente de ver cómo "funcionaría" una sola clase que tenga varios atributos y métodos.

    - Ejemplo de clase integradora (**IMPORTANTE: tómense su tiempo para analizar el ejemplo, tengan ambas clases "Libro" y "Biblioteca" a mano para poder seguir las acciones**):

```
//clase de integración o de "test"
//la usamos para "probar" y cómo se usarían nuestras clases
Clase Biblioteca
    //constructor
    metodo Biblioteca()
        //invoco método locales
        crearLibroUsandoMetodos()
        crearLibroAccediendoAtributos()
    fin metodo

    metodo crearLibroUsandoMetodos()
        //creo un atributo local
        Libro seniorDeLosAnillos nuevaInstancia Libro()

        //utilizo el objeto creado + operador punto "." + método getter
        //para acceder al valor que retorna el método
        mostrar: seniorDeLosAnillos.obtenerAutor()  //muestra "Anónimo"

        //utilizo el objeto creado + operador punto "." + método setter
        //para modificar el valor del atributo
        seniorDeLosAnillos.asignarAutor("JRR Tolkien")

        //repito el uso del objeto creado + operador punto "." + método getter
        //para acceder al valor que retorna el método
        mostrar: seniorDeLosAnillos.obtenerAutor()  //muestra  "JRR Tolkien"
    fin metodo

    metodo crearLibroAccediendoAtributos()
        //creo objeto (atributo) local
        Libro laTorreOscura nuevaInstancia Libro()

        //utilizo el objeto creado + operador punto "." + atributo
        //para modificar el valor del atributo
        laTorreOscura.autor = "Stephen King"

        //utilizo el objeto creado + operador punto "." + atributo para acceder
        //al valor del atributo
        mostrar: laTorreOscura.autor    //muestra "Stephen King"
    fin metodo
fin clase
```

- Formas de acceder a los datos (atributos):
  - En "crearLibroUsandoMetodos()" accedo a los atributos del objeto utilizando los métodos setter y getter (es la forma recomendada).
  - En "crearLibroAccediendoAtributos()" accedo a los atributos del objeto en forma directa.
  - Ambas alternativas son válidas, pero se recomienda la primera por conceptos que veremos en la unidad 10.

- Ahora un ejemplo un poco más complicado para que analicen...

```
Clase Biblioteca
    //cada elemento del vector va a guardar un objeto del tipo "Libro"
    //declaro un atributo que será un vector de 10 posiciones
    Libro catalogo[10] nuevaInstancia Libro()

    //constructor
    metodo Biblioteca()
        //agrego 3 películas a mi catálogo de libros...
        catalogo[1] = crearLibro("El señor de los anillos", "JRR Tolkien", 1954)
        catalogo[2] = crearLibro("IT", "Stephen King", 1986)
        catalogo[3] = crearLibro("Dune", "Frank Herbert", 1965)

        //muestro la antiguedad de los 3 libros cargados
        mostrar: catalogo[1].calcularAntiguedad()   //2020-1954 = 66: muestra "66"
        mostrar: catalogo[2].calcularAntiguedad()   //2020-1986 = 34: muestra "34"
        mostrar: catalogo[3].calcularAntiguedad()   //2020-1965 = 55: muestra "55"
        //Tip: cada elemento de vector contiene un objeto, por lo que se usa el operador "."
            //para invocar sus métodos, como se haría con cualquier objeto
    fin metodo

    metodo Libro crearLibro(String nombre, String autor, Integer anio)
        Libro libroNuevo nuevaInstancia Libro()

        libroNuevo.asignarNombre(nombre)
        libroNuevo.asignarAutor(autor)
        libroNuevo.asignarAnioPublicacion(anio)

        retornar: libroNuevo
    fin metodo
fin clase
```

- Pregunta frecuente, referida a los ejemplos de las páginas 12 a 15 (Auto, Color y Marca): "*¿Por qué el método definir color/marca se declara tanto en esas clases Color y Marca como en la clase auto?*"
    - Primero: es una coincidencia que tengan el mismo nombre ("definirColor" y "definirMarca"). Y es para demostrar que es perfectamente válido. Cada objeto es una entidad independiente, que puede tener métodos y atributos. Por lo tanto, vemos que en la clase Auto tenemos un método "definirColor" que recibe como parámetro un objeto del tipo Color. Este método lo va a usar el objeto Auto, ya que "sabe" como cambiarse de color.
    - Ahora bien, ya dimos por supuesto que existen ciertas clases que no vamos a tener que definir, las damos por "dadas" en el pseudocódigo, como sucede en todos los lenguajes (String, Boolean, Integer, etc).
    - Esto no ocurre con las clases "de negocio" que son las clases que define el programador. Entre esas tenemos la clase Color, que termina encapsulando el dato del color como un String, pero "envuelto" en una clase propia. Y como el "color sabe cambiarse de color", también tiene un método "definirColor" (pero en este caso recibe un String, porque es el dato que guarda).
    - Entonces, la idea, en narrativo, sería:
        - creo un objeto color
        - guardo el dato del color en ese objeto en forma de objeto tipo string
        - creo un objeto auto
        - guardo el dato del color del auto en forma de objeto tipo color

ACTIVIDAD DE ANÁLISIS:

¿Qué otros elementos que tienen cerca en este momento podrían pensar en términos de clases y objetos? Pensar y modelar algún elemento que elijan, luego definir una clase y, finalmente, instanciarla.

Para esto, se deberán seguir los siguientes lineamientos:

- Contar con 1 clase que modela el elemento elegido.
- Dicha clase debe tener como mínimo 1 atributo, con su setter y getter.
- Además, deberá tener como mínimo 1 método que haga una o varias operaciones (un cálculo, una asignación, ingreso de datos, etc.). Este método puede recibir parámetros y devolver un valor.
- Además, debe haber una clase adicional de integración o test que genere 1 o 2 instancias diferentes de la clase previamente definida
- Todas las clases deberán tener sus respectivos constructores

Podrán comparar sus respuestas con los ejemplos resueltos en el "ANEXO 1 - Respuesta Actividad de Análisis" al final de esta unidad.

Previamente pueden consultar el "ANEXO 2 - Checklist de código".

IMPORTANTE:

- **¡Traten de resolverlo cada uno por sus propios medios!**
- **Las respuestas (el programa) NO debe ser enviado a los foros del Campus, salvo dudas puntuales de las respuestas.**

ANEXO 1 - Respuesta Actividad de Análisis

¿Trataron de resolver los requerimientos?

En caso afirmativo, avancen 3 páginas.

En caso negativo, ¡no avancen y traten de resolverlos antes de ver las respuestas!

Página dejada intencionalmente en blanco

Página dejada intencionalmente en blanco

Página dejada intencionalmente en blanco

1)

```
clase Teclado
    String color nuevaInstancia String()
    String numerico nuevaInstancia String()

    metodo Teclado()
    fin metodo

    metodo asignarColor(String nuevoCol)
        color = nuevoCol
    fin metodo

    metodo String obtenerColor()
        retornar: color
    fin metodo

    metodo String descubrirNumerico(Integer nuevaT)
        si nuevaT < 100 entonces
            numerico = "no tiene teclado numerico"
        sino
            numerico = "tiene teclado numerico"
        fin si
        retornar: numerico
    fin metodo
fin clase

clase TestTeclado
    metodo TestTeclado()
        Teclado teclado1 nuevaInstancia Teclado()
        Teclado teclado2 nuevaInstancia Teclado()

        teclado1.asignarColor("negro")
        mostrar: teclado1.obtenerColor()
        mostrar: teclado1.descubrirNumerico(105)
        mostrar: teclado1.color

        teclado2.asignarColor("blanco")
        mostrar: teclado2.obtenerColor()
        teclado2.descubrirNumerico(98)
        mostrar: teclado2.color
    fin metodo
fin clase
```

2)

```
clase Termo
    Float capacidadDelTermo nuevaInstancia Float ()

    metodo Termo()
        capacidadDelTermo = 1
    fin metodo

    metodo definirCapacidadDelTermo (Float capacidadNueva)
        capacidadDelTermo = capacidadNueva
    fin metodo

    metodo Float devolverCapacidadDelTermo ()
        retornar: capacidadDelTermo
    fin metodo

    metodo Boolean seLlena (float cantidadLiquido)
        si capacidadDelTermo > cantidadLiquido entonces
            retornar: falso
        sino
            retornar: verdadero
        fin si
    fin metodo
fin clase

clase Cocina // clase Integración o Test
    metodo Cocina()
        crearTermoYLlamarMetodo()
    fin metodo

    metodo crearTermoYLlamarMetodo()
        Termo termo nuevaInstancia Termo()
        termo.definirCapacidadDelTermo(1.0)
        mostrar: termo.seLlena(2) // muestra verdadero
        mostrar: termo.seLlena(0.5) // muestra falso
        mostrar: termo.seLlena(1.0) // muestra verdadero
        termo.devolverCapacidadDelTermo() //devuelve 1
    fin metodo
fin clase
```

3)

```
clase Auricular
    String marca nuevaInstancia String()
    String formato nuevaInstancia String()
    Float cantidad nuevaInstancia Float()
    Float costo nuevaInstancia Float()
    Float valorTotal nuevaInstancia Float()

    metodo Auricular() // metodo constructor
    fin metodo

    metodo ingresarMarca(String marcaRecibida)
        marca = marcaRecibida
    fin metodo

    metodo ingresarFormato(String formatoRecibido)
        formato = formatoRecibido
    fin metodo

    metodo ingresarUnidadesYelImporte(Float cantidadRecibida, Float costoRecibido)
        cantidad = cantidadRecibida
        costo = costoRecibido
    fin metodo

    metodo String devolverMarca()
        retornar: marca
    fin metodo

    metodo String devolverFormato()
        retornar: formato
    fin metodo

    metodo Float devolverPrecioDeVenta()
        valorTotal = 1.5 * (costo / cantidad)
        retornar: valorTotal
    fin metodo
fin clase
```

```
clase RegistrarTest
    metodo RegistrarTest()// metodo constructor
        Auricular productoA nuevaInstancia Auricular()
        Auricular productoB nuevaInstancia Auricular()

        productoA.ingresarMarca("SONY")
        productoB.ingresarMarca("JBL")

        productoA.ingresarFormato("Heatset")
        productoB.ingresarFormato("In-ear")

        productoA.ingresarUnidadesYelImporte(10.0, 70000.0)
        productoB.ingresarUnidadesYelImporte(20.0, 40000.0)

        mostrar: productoA.devolverMarca()
        mostrar: productoA.devolverFormato()
        mostrar: productoA.devolverPrecioDeVenta()

        mostrar: productoB.devolverMarca()
        mostrar: productoB.devolverFormato()
        mostrar: productoB.devolverPrecioDeVenta()
    fin metodo
fin Clase
```

4)

```
clase Birome
    //atributos
    String color nuevaInstancia String()

    //constructor
    metodo Birome()
        color = ""
    fin metodo

    //setter
    metodo asignarElColor(String colorNuevo)
        color = colorNuevo
    fin metodo

    //getter
    metodo String mostrarElColor()
        retornar: color
    fin metodo

fin clase

clase BiromeTest

    metodo BiromeTest() //defino las instancias dentro
    //del método constructor de la clase.
        Birome color1 nuevaInstancia Birome()
        Birome color2 nuevaInstancia Birome()
        String colorAux nuevaInstancia String()

        mostrar: "Ingresar el primer color:"
        ingresar: colorAux
        color1.asignarElColor(colorAux)
        mostrar: "Seleccionado color:" + color1.mostrarElColor()

        mostrar: "Ingresar el segundo color:"
        ingresar: colorAux
        color2.asignarElColor(colorAux)
        mostrar: "Seleccionado color:" + color2.mostrarElColor()

    fin metodo

fin clase
```

5)

```
clase Notebook
    String modelo nuevaInstancia String()
    Float valorVenta nuevaInstancia Float()
    Float valorCosto nuevaInstancia Float()

    metodo Notebook()
        valorCosto = 80000.0
    fin metodo

    metodo asignarValorCosto (Float v)
        valorCosto = v
    fin metodo

    // Getter: setea el modelo
    metodo asignarModelo(String modelo)
        nuevoModelo = modelo
    fin metodo

    // Getter: obtiene el modelo asignado
    metodo String obtenerModelo()
        retornar: modelo
    fin metodo

    // Metodo para realizar calculo
    metodo calculoValorVenta()
        valorVenta = valorCosto * 1.3
        mostrar: valorVenta
    fin metodo
fin clase

// clase de Integracion o Test
clase NotebookTest

    metodo NotebookTest()
        Notebook modelo1 nuevaInstancia Notebook()
        Notebook modelo2 nuevaInstancia Notebook()

        modelo1.asignarModelo("Gamer")
        //pisamos el valor por defecto asignado en el constructor
        modelo1.asignarValorCosto(90000.00)

        modelo2.asignarModelo("Diseño")

        modelo1.calculoValorVenta()
        modelo2.calculoValorVenta()
    fin metodo
fin clase
```

6)

```
clase Socio
    String nombre nuevaInstancia String()

    metodo Socio()
        nombre = ""
    fin metodo

    metodo asignarNombre (String nombreNuevo)
        si nombreNuevo = "" entonces
            ingresar: nombre
        sino
            nombre = nombreNuevo
        fin si
    fin metodo

    metodo String devolverNombre()
        retornar: nombre
    fin metodo
fin clase

clase ClubTest
    metodo ClubTest()
        Socio nombre1 nuevaInstancia Socio()
        Socio nombre2 nuevaInstancia Socio()

        nombre1.asignarNombre("Marcelo Daniel G")
        nombre2.asignarNombre("")

        mostrar: nombre1.devolverNombre()  //muestra "Marcelo Daniel G"
        mostrar: nombre2.devolverNombre()  //muestra lo que ingresa el usuario
    fin metodo
fin clase
```

7)

```
clase Nota
    Integer nota nuevaInstancia Integer()

    metodo definirNota(Integer nuevaNota)
        nota = nuevaNota
    fin metodo

    metodo Integer devolverCondicion()
        si nota >= 7 Y nota <= 10 entonces
            mostrar: "aprobado/a"
            retornar: nota
        sino
            si nota > 0 Y nota < 7 entonces
                mostrar: "desaprobado/a"
                retornar: nota
            sino
                mostrar: "nota no válida"
                retornar: nota
            fin si
        fin si
    fin metodo
fin clase

clase NotaTest
    metodo alumnoTest()
        Nota nota nuevaInstancia nota()

        nota.definirNota(5)
        //muestra desaprobado/a
        mostrar: nota.devolverCondicion()

        nota.definirNota(14)
        //muestra nota no válida
        mostrar: nota.devolverCondicion()

        nota.definirNota(8)
        //muestra aprobado/a
        mostrar: nota.devolverCondicion()
    fin metodo
fin clase
```

8)

```
clase Flores
    String nombreDeFlor nuevaInstancia String()
    Integer toxicidad nuevaInstancia Integer()

    metodo asignarNombredeFlor(String nombrenuevoDeFlor)
        nombreDeFlor = nombrenuevoDeFlor
    fin metodo

    metodo Integer definirToxicidad()
        mostrar: "Ingrese nivel de toxicidad de la flor en una escala de 1 a 10"
        ingresar: toxicidad
        si toxicidad <= 0 O toxicidad > 10 entonces
            mostrar: "el valor ingresado no es correcto"
            definirToxicidad()
        fin si
        retornar: toxicidad
    fin metodo

    metodo sobreToxicidad()
        si toxicidad <= 5 entonces
            mostrar: "La flor NO es peligrosa para su mascota"
        sino
            mostrar: "La flor ES peligrosa para su mascota"
        fin si
    fin metodo

    metodo Flores()
    fin metodo
fin clase

clase FloresTest
    metodo FloresTest()
        Flores flor1 nuevaInstancia Flores()
        Flores flor2 nuevaInstancia Flores()

        flor1.asignarNombredeFlor("azalea")
        flor1.definirToxicidad()
        flor1.sobreToxicidad()

        flor2.asignarNombredeFlor("tulipán")
        flor2.definirToxicidad()
        flor2.sobreToxicidad()
    fin metodo
fin clase
```

9)

```
clase Perro
    String nombre nuevaInstancia String()
    Integer edad nuevaInstancia Integer()
    Integer anioNacimiento nuevaInstancia Integer()
    Integer anioActual nuevaInstancia Integer()
    String raza nuevaInstancia String()
    String genero nuevaInstancia String()

    metodo Perro()
    fin metodo

    metodo asignarRaza(String razaNueva)
        raza = razaNueva
    fin metodo

    metodo asignarNombre(String nombreNuevo)
        nombre = nombreNuevo
    fin metodo

    metodo asignarAnioNacimiento(Integer anioNacimientoNuevo)
        anioNacimiento = anioNacimientoNuevo
    fin metodo

    metodo asignarAnioActual(Integer anioActualNuevo)
        anioActual = anioActualNuevo

    metodo asignarGenero(String generoNuevo)
        genero = generoNuevo
    fin metodo

    metodo calcularEdad()
        edad = anioActual - anioNacimiento
    fin metodo

    metodo String obtenerRaza()
        retornar: raza
    fin metodo

    metodo String obtenerNombre()
        retornar: nombre
    fin metodo

    metodo Integer obtenerEdad()
        retornar: edad
    fin metodo
```

```
    metodo Integer obtenerAnioNacimiento()
        retornar: anioNacimiento
    fin metodo

    metodo String obtenerGenero()
        retornar: genero
    fin metodo

fin clase

clase PerrosTest
    Perro perroPrimero nuevaInstancia Perro()
    Perro perroSegundo nuevaInstancia Perro()

    metodo PerrosTest
        ingresarPerro()
        obtenerPerro()
    fin metodo

    metodo ingresarPerro()
        perroPrimero.asignarNombre("Terry")
        perroPrimero.asignarRaza("Ovejero Aleman")
        perroPrimero.asignarAnioNacimiento(2017)
        perroPrimero.asignarGenero("Masculino")
        perroPrimero.asignarAnioActual(2020)
        perroPrimero.calcularEdad()

        perroSegundo.asignarNombre("Violeta")
        perroSegundo.asignarRaza("Mestizo")
        perroSegundo.asignarAnioNacimiento(2014)
        perroSegundo.asignarGenero("Femenino")
        perroSegundo.asignarAnioActual(2020)
        perroSegundo.calcularEdad()
    fin metodo

    metodo obtenerPerro()
        mostrar: perroPrimero.obtenerNombre()
        mostrar: perroPrimero.obtenerRaza()
        mostrar: perroPrimero.obtenerAnioNacimiento()
        mostrar: perroPrimero.obtenerEdad()
        mostrar: perroPrimero.obtenerGenero()

        mostrar: perroSegundo.obtenerNombre()
        mostrar: perroSegundo.obtenerRaza()
        mostrar: perroSegundo.obtenerAnioNacimiento()
        mostrar: perroSegundo.obtenerEdad()
        mostrar: perroSegundo.obtenerGenero()
    fin metodo
fin clase
```

10)

```
clase Cerveza
    String tipo nuevaInstancia String()

    metodo Cerveza()
    fin metodo

    metodo String obtenerTipo()
        retornar: tipo
    fin metodo

    metodo determinarTipo(String t)
        si t = "" entonces
            mostrar: "Ingrese tipo de la cerveza"
            ingresar: tipo
        sino
            tipo = t
            mostrar: "El tipo de cerveza ya se ha modificado"
        fin si
    fin metodo
fin clase

clase CervezaTest
    metodo CervezaTest()
        Cerveza cerveza1 nuevaInstancia Cerveza()
        Cerveza cerveza2 nuevaInstancia Cerveza()

        cerveza1.determinarTipo("Amber") //muestra "El tipo de cerveza ya se ha modificado"
        mostrar: cerveza1.obtenerTipo()  //muestra Amber

        cerveza2.determinarTipo("")
        mostrar: cerveza2.obtenerTipo()  //muestra lo ingresado por el usuario
    fin metodo
fin clase
```

11)

```
clase Monitor

    String marca nuevaInstancia String()
    Float pulgadas nuevaInstancia Float()
    String gamer nuevaInstancia String()

    metodo Monitor()
    fin metodo

    metodo definirMarca(String marcaCliente)
        marca = marcaCliente
    fin metodo

    metodo definirPulgadas(Float pulgadasCliente)
        pulgadas = pulgadasCliente
    fin metodo

    metodo String gamerOno()
        si pulgadas >= 27.0 entonces
            mostrar: "Hola tu monitor es gamer! tambien se puede usar para el competitivo!"
            gamer = "si"
        sino
            mostrar: "Hola tu monitor es estandar, pero tambien puedes jugar!"
            gamer = "no"
        fin si
        retornar: gamer
    fin metodo

fin clase

clase MonitorTest
    metodo MonitorTest()
        Monitor monitor1 nuevaInstancia Monitor()
        Monitor monitor2 nuevaInstancia Monitor()

        monitor1.definirMarca("aoc")
        monitor1.definirPulgadas(22.1)
        mostrar: monitor1.gamerOno()

        monitor2.definirMarca("lg")
        monitor2.definirPulgadas(24.7)
        mostrar: monitor2.gamerOno()
    fin metodo
fin clase
```

12)

```
clase Celular
    String modelo nuevaInstancia String()
    Integer cantidad nuevaInstancia Integer()

    metodo Celular()
        cantidad = 10
        modelo = "StarTAC"
    fin metodo

    metodo Boolean vender()
        si cantidad > 0 entonces
            cantidad = cantidad - 1
            retornar: verdadero
        sino
            mostrar: "Celular sin stock"
            retornar: falso
        fin si
    fin metodo

    metodo String obtenerModelo()
        retornar: modelo
    fin metodo
fin clase

clase TestCelular
    metodo TestCelular()
        Celular celu nuevaInstancia Celular()

        mientras celu.vender() = verdadero hacer
            mostrar: "se vendió un " + celu.obtenerModelo()
                + " y quedan en stock " + celu.cantidad
        fin mientras
    fin metodo
fin clase
```

13)

```
clase Clima
    Integer humedadRelativa nuevaInstancia Integer()
    String pronostico nuevaInstancia String()

    metodo Clima()
        humedadRelativa = 0
        pronostico = ""
    fin metodo

    metodo recibirPronostico (Integer humedadRelativaActual)
        humedadRelativa = humedadRelativaActual
        si humedadRelativa <= 50 entonces
            pronostico = "Baja probabilidad de lluvia"
        sino
            si humedadRelativa >= 51 Y humedadRelativa <= 80 entonces
                pronostico = "Moderada probabilidad de lluvia"
            sino
                pronostico = "Alta probabilidad de lluvia"
            fin si
        fin si
    fin metodo

    metodo String enviarPronostico()
        retornar: pronostico
    fin metodo
fin clase

clase ObtenerPronostico
    metodo ObtenerPronostico()
        Clima verPronostico nuevaInstancia Clima()
        Integer humedadDeHoy nuevaInstancia Integer()

        mostrar: "Ingrese la humedad relativa de este momento"
        ingresar: humedadDeHoy

        verPronostico.recibirPronostico(humedadDeHoy)
        mostrar: verPronostico.enviarPronostico()
    fin metodo
fin clase
```

14)

```
clase PlacaVideo
    String marcaPlacaVideo nuevaInstancia String()
    String modeloPlacaVideo nuevaInstancia String()

    metodo PlacaVideo()
    fin metodo

    metodo String obtenerMarca()
        retornar: marcaPlacaVideo
    fin metodo

    metodo String obtenerModelo()
        retornar: modeloPlacaVideo
    fin metodo

    metodo asignarMarca(String marcaNueva)
        marcaPlacaVideo = marcaNueva
    fin metodo

    metodo asignarModelo(String modeloNuevo)
        modeloPlacaVideo = modeloNuevo
    fin metodo
fin clase

clase Gpu    //test
    metodo Gpu()
        PlacaVideo gpuA1 nuevaInstancia PlacaVideo()
        PlacaVideo gpuB1 nuevaInstancia PlacaVideo()

        gpuA1.asignarMarca("AMD")
        gpuB1.asignarMarca("NVIDIA")

        mostrar: gpuA1.obtenerMarca()
        mostrar: gpuB1.obtenerMarca()
    fin metodo
fin clase
```

15)

```
clase Radio
    // atributo
    String nombreEstacion nuevaInstancia String()

    // constructor default
    metodo Radio()
    fin metodo

    // método getter
    metodo String obtenerEstacion()
        retornar:  nombreEstacion
    fin metodo

    // método setter
    metodo asignarEstacion(String estacionNueva)
        nombreEstacion = estacionNueva
    fin metodo

    metodo mostrarEstacion(String nombreEstacion)
        mostrar: nombreEstacion
    fin metodo
fin clase

clase RadioTest
    metodo RadioTest()
        String estacion nuevaInstancia String()
        Radio radio1 nuevaInstancia Radio()
        Radio radio2 nuevaInstancia Radio()

        radio1.asignarEstacion("Radio Nacional")
        estacion = radio1.obtenerEstacion()
        radio1.mostrarEstacion(estacion)

        radio2.asignarEstacion("Radio Romántica")
        radio2.mostrarEstacion(radio2.obtenerEstacion())
    fin metodo
fin clase
```

16)

```
clase Bicicleta
    String nombre nuevaInstancia String()
    String local nuevaInstacia String()

    metodo Bicleta()
        mostrar: "Ingrese la marca de su bicicleta"
        ingresar: nombre
        local = "Nuñez"
    fin metodo

    metodo asignarMarca(String marcaNueva)
        nombre = marcaNueva
    fin metodo

    metodo String obtenerDatos()
        retornar: "bicicleta " + nombre + " /// se vende en " + local
    fin metodo
fin clase

clase TestBici
    metodo TestBici()
        Bicleta bici nuevaInstancia Bicleta()
        bici.asignarMarca("Zenith")
        mostrar: bici.obtenerDatos()
    fin metodo
fin clase
```

17)

```
clase Paniales
    String marcaPanial nuevaInstancia String()

    metodo Paniales()
    fin metodo

    metodo setMarca(String marca)
        marcaPanial = marca
    fin metodo

    metodo String getMarca()
        retornar: marcaPanial
    fin metodo

    metodo String elegirMarcaPanial(String marca)
        si marca = "Huggies" entonces
            retornar: "Active Sec"
        sino si marca = "Pampers" entonces
            retornar: "SuperSec"
        sino
            retornar: "ERROR"
        fin si
    fin metodo
fin clase

clase PanialesTest
    metodo PanialesTest()
        Panial panial1 nuevaInstancia Panial()
        Panial panial2 nuevaInstancia Panial()

        panial1.setMarca("Huggies")
        panial2.setMarca("Pampers")

        mostrar: panial1.elegirMarcaPanial(panial1.getMarca())
        mostrar: panial2.elegirMarcaPanial(panial2.getMarca())
    fin metodo
fin clase
```

18)

```
clase Electrodomestico
    String marcaDelElectrodomestico nuevaInstancia String()

    metodo Electrodomestico()
    fin metodo

    metodo asignarMarca(String marca)
        marcaDelElectrodomestico = marca
    fin metodo

    metodo String obtenerMarca()
        retornar: marcaDelElectrodomestico
    fin metodo

    metodo Integer aniosDeGarantia()
        si marcaDelElectrodomestico = "Whirpool" entonces
            retornar: 3
        sino si marcaDelElectrodomestico = "Samsung" entonces
            retornar: 2
        sino
            retornar: 1
        fin si
    fin metodo
fin clase

clase ElectrodomesticoTest
    metodo ElectrodomesticoTest()
        crearInstanciaDeElectrodomestico()
    fin metodo

    metodo crearInstanciaDeElectrodomestico()
        Electrodomestico heladera nuevaInstancia Electrodomestico()
        Electrodomestico microondas nuevaInstancia Electrodomestico()

        heladera.asignarMarca("Whirpool")
        mostrar: "tiene " + heladera.aniosdeGarantia() + " años de garantía"

        microondas.asignarMarca("Samsung")
        mostrar: "tiene " + microondas.aniosdeGarantia() + " años de garantía"
    fin metodo
fin clase
```

19)

```
clase Embarcacion
    String embarcacion nuevaInstancia String()

    metodo Embarcacion()
        embarcacion = "S/D"
    fin metodo

    metodo asignarNombre (String nuevoNombre)
        embarcacion = nuevoNombre
    fin metodo

    metodo String obtenerNombre()
        retornar: embarcacion
    fin metodo
fin clase

clase Creacion
    metodo Creacion()
        nuevaEmbarcacion()
    fin metodo

    metodo nuevaEmbarcacion()
        Embarcacion embarcacionLolita nuevaInstancia Embarcacion()
        Embarcacion embarcacionMarAzul nuevaInstancia Embarcacion()

        embarcacionLolita.asignarNombre("lolita")
        mostrar: "Embarcacion:" + embarcacionLolita.obtenerNombre()

        embarcacionMarAzul.asignarNombre("mar azul")
        mostrar: "Embarcacion" + embarcacionMarAzul.obtenerNombre()
    fin metodo
fin clase
```

20)

```
clase MountainBike
    String modelo nuevaInstancia String()

    metodo MountainBike()
        modelo = ""
    fin metodo

    metodo String obtenerModelo()
        retornar: modelo
    fin metodo

    metodo asignarModelo(String modeloNuevo)
        si modeloNuevo != "" entonces
            modelo = modeloNuevo
        sino
            mostrar: "por favor ingrese modelo"
            ingresar: modelo
        fin si
    fin metodo
fin clase

clase MountainBikeTest
    metodo MountainBikeTest()
        MountainBike mountain1 nuevaInstancia MountainBike()
        MountainBike mountain2 nuevaInstancia MountainBike()

        mountain1.asignarModelo("")
        mountain2.asignarModelo("scott voltage yz20")

        mostrar: mountain1.modelo
        mostrar: mountain2.modelo
    fin metodo
fin clase
```

ANEXO 2 - Checklist de código

Algunas consideraciones a la hora de revisar su propio código que se suman al checklist de la unidad anterior:

	Cumple	
Tema	**Si**	**No**
Ya no se utlizan las palabras reservadas “programa”, “inicio”, “fin”, “funcion”, “fin funcion”, “var” y “const”		
Los métodos cumplen con las mismas reglas que las funciones (en cuanto a parámetros y valores devueltos)		
Se utilizan las palabras reservadas “clase” y “fin clase” y “metodo” “fin metodo”		
Todas las clases tienen un método constructor		
Todos los atributos se declaran usando la fórmula: CLASE objeto PALABRA_RESERVADA_NUEVA_INSTANCIA CONSTRUCTOR()		
Una clase que tiene una invocación a un método que se encuentra declarado en otra clase lo está invocando a través de un objeto y el operador punto. Ejemplo: otraClase.elMetodo()		
Los métodos getter no reciben parámetro, devuelve un objeto y solo hacen “retonar: obj”		
Los métodos setter reciben un parámetro, no devuelve valores y solo hacen “atributo = parámetro”		
Una clase que invoca un método que está declarado en esa misma clase, hace la invocación en forma directa (sin usar un objeto). Por ejemplo: “miMetodoInterno()”		
Todas los métodos invocados a través de un objeto y el operador punto se encuentran declarados en la clase a partir de la cual se creó el objeto. Por ejemplo, en la invocación: otraClase.elMetodo() Estamos seguros que el método “elMetodo()” se encuentra en el clase de la cual es instancia el objeto “otraClase”. Teniendo: LaClase otraClase nuevaInstancia LaClase() El método “elMetodo()” EXISTE en la clase “LaClase”.		
Todos los objetos utilizados existen: fueron declarados como atributos o son parámetros de un método		

Si todas las respuestas son afirmativas, hay muy altas chances que el programa esté libre de errores de código.

ANEXO 3 - Preguntas frecuentes

1) Pregunta: Hay clase que se llama Club con una instrucción, la "instancia de una clase", donde decimos que el primer "String" que es la clase a partir de la cual se crea el objeto, pero, ¿cómo se relaciona ese "String" con la clase "Club"?

Respuesta: En el ejempplo de la clase Club tenemos un objeto llamado "nombre", que es instancia de la clase "String". A su vez, ese objeto es un atributo de la clase "Club".

Cuando sea necesario crear un club nuevo, se va a poder usar ese atributo para darle un nombre. Por ejemplo:

```
Club ponechispas nuevaInstancia Club()

ponechispas.nombre = "Club Social y Deportivo Ponechispas"
```

2) P: No queda en claro para qué sirven los objetos auxiliares en el ejemplo del Auto, Color y Marca: "auxColor", "auxMarca", etc.

R: Esos objetos no tienen utilidad práctica, más allá de demostrar cómo se hacen asignaciones usando objetos (que, de hecho, sería de la misma manera que se hace usando variables).

3) P: A partir de la frase "en objetos, todo es objetos", la pregunta es ¿un método también es un objeto?, porque un atributo también se lo puede que ver como objeto...

R: No, un método es un método. La frase "en objetos, todo es objetos" se refiere a los atributos (antes "variables").

Un objeto, ¿qué es? Es una instancia de una clase. ¿Qué tiene una clase? Métodos y atributos. ¿Qué es un método? El mecanismo que nos permite darle comportamiento al objeto. ¿Qué es un atributo? El mecanismo que no permite crear un objeto. Un objeto, ¿qué es? Una instancia de una... (loop infinito).

4) P: Con respecto a la invocación del método "definirColor" en la clase "EjemploAuto": cuando se crea el objeto "primerAuto", se declara además un atributo "primerAutoColor" al que luego se le asigna el color "gris plata". ¿Por qué ese atributo "primerAutoColor" se usa para invocar al método "difinirColor", cuando este método se encuentra en la clase "Auto"? ¿No se tendría que invocar de la siguiente manera:?

primerAuto.definirColor("gris plata")

```
clase EjemploAuto
    metodo EjemploAuto()
        //primer objeto de tipo Auto creado
        Auto primerAuto nuevaInstancia Auto()

        Color primerAutoColor nuevaInstancia Color()
        Color auxColor nuevaInstancia Color()
        Marca primerAutoMarca nuevaInstancia Marca()
        Marca auxMarca nuevaInstancia Marca()

        primerAutoColor.definirColor("gris plata")
        auxColor = primerAutoColor
        primerAutoMarca.definirMarca("VW")
        auxMarca = primerAutoMarca

        primerAuto.definirColor(auxColor)
        primerAuto.definirMarca(auxMarca)

        //segundo objeto de tipo Auto creado
        Auto segundoAuto nuevaInstancia Auto()
        Color segundoAutoColor nuevaInstancia Color()
        Marca segundoAutoMarca nuevaInstancia Marca()

        segundoAutoColor.definirColor("scandium")
        segundoAuto.definirColor(segundoAutoColor)

        segundoAutoMarca.definirMarca("Fiat")
        segundoAuto.definirMarca(segundoAutoMarca)

    fin metodo
fin clase
```

R: No. Hay que prestar atención a las clases Auto y Color: ambas tienen un método llamado "definirColor" (un setter en ambos casos) y son métodos independientes en clases independientes. El detalle es que el método "definirColor" de la clase "Auto" recibe como parámetro un objeto del tipo "Color". Por otro lado, el método "definirColor" de la clase "Color" recibe como parámetro un objeto de tipo "String".

Dicho esto:

primerAuto.definirColor(ACA VA UN OBJETO COLOR)

primerAutoColor.definirColor(ACA VA UN OBJETO STRING)

Por esto es que se hace:

primerAuto.definirColor(auxColor)

Porque "auxColor" es un objeto del tipo Color, que se le asignó el objeto "primerAutoColor".

La narrativa sería: "primero se crea un objeto tipo Color, al cual se le asigna un String para darle valor. Luego se usa ese objeto tipo Color para asignárselo al objeto tipo Auto".

Es importante, en las invocaciones de los métodos, ver qué objeto se está utilizando, porque ese método deberá estar declarado en la clase que se instancia para crear dicho objeto. Dicho de otra manera hay una clara trazabilidad que se mantiene con respecto a:

método invocado

⇩ (ver)

objeto que se usa en la invocación

⇩ (ver)

clase que se para crear el objeto

⇩ (debe tener)

el método invocado

5) P: ¿Cómo podría usar el tipo Boolean en un ejemplo con clases, atributos y métodos?

R: Un ejemplo del uso de objeto tipo Boolean en una clase podría ser:

```
clase Validacion
    metodo  Validacion()
    fin metodo

    metodo Boolean validarDNI(Integer nro, Boolean esArgentino)
        si esArgentino = verdadero entonces
            si nro > 0 Y nro < 100,000,000 entonces
                retornar: verdadero
            sino
                retornar: falso
            fin si
        sino
            retornar: falso
        fin si
    fin metodo
fin clase


clase CrearDNI //test
    metodo  CrearDNI()
        Boolean rdo nuevaInstancia Boolean()
        Validacion val nuevaInstancia Validacion()

        rdo = val.validarDNI(26485788, verdadero)
        si rdo = verdadero entonces
            mostrar: "este es correcto"     // sale por acá!
        sino
            mostrar: "este NO es correcto"
        fin si

        rdo = val.validarDNI(226221521, verdadero)
        si rdo = verdadero entonces
            mostrar: "este es correcto"
        sino
            mostrar: "este NO es correcto"       // sale por acá!
        fin si

        rdo = val.validarDNI(26485788 , falso)
        si rdo = verdadero entonces
            mostrar: "este es correcto"
        sino
            mostrar: "este NO es correcto"     // sale por acá!
        fin si
    fin metodo
fin clase
```

6) P: Sobre la frase "Los constructores se ejecutan siempre que se crea un objeto y se utiliza cuando se instancia una clase": ¿Esto significa que si se crea más de un objeto a partir de una clase y, por ejemplo, dentro del constructor hay un ciclo "mientras", este se va a ejecutar cada vez que cree un objeto de esa clase?

R: Si, exactamente eso. Hay que tener en cuenta que el ciclo se va a ejecutar para los diferentes objetos que vayan creando, por lo tanto son "diferentes instancias ("ejecuciones") del mismo ciclo"

7) P: En el momento de declarar los atributos ¿por qué en algunos de los ejemplos se los declara siguiendo la formula :

String nombre nuevaInstancia String()

Y en otros, por ejemplo:

Auto nombre nueva instancia Auto()

¿Es porque en el segundo ya fue creada una clase anteriormente denominada "Auto"?

R: Así como en Estructurado se dio por hecho que el pseudocódigo (como los lenguajes) provee de los tipos de datos, lo mismo ocurre en Objetos: asumimos que ya tenemos ciertos tipos de datos, pero que son clases al mismo tiempo.

De hecho, cualquier clase que se instancia es un "tipo de dato", en el sentido en que se están creando objetos a partir de esa clase. Por lo que si tenemos:

A x nuevaInstancia A()

Decimos: "el objeto "x" es del tipo de la clase A".

8) P: ¿Los objetos están dentro de una clase?

R: Los objetos se declaran en las clases (en los atributos), en tiempo de programación. A su vez, son instancias de las clases, en tiempo de ejecución.

9) P: Es válida la analogía "los objetos son lo que eran (en estructurado) las variables?

R: Si, pero además de poder guardar varios datos y no uno solo (que son los atributos), como si fuera poco, tienen comportamiento (que son los métodos).

10) P: ¿Los atributos son lo que eran las variables (en estructurado)?

R: Ídem respuesta anterior

11) P: ¿Los objetos tienen atributos?

R: Las clases tienen atributos, los cuales se pueden usar una vez que se crear una instancia de esa clase.

12) P: ¿Se puede hacer una clase a partir de un objeto creado en la clase que antes lo contenía?

R: No, no se puede ir de clase a objeto. Pero sí al revés: a partir de una clase se crea (instancia) un objeto.

13) P: Dentro de un método se le asigna el valor del parámetro "marcaNueva" al atributo de la clase "Marca". ¿Qué es "marca" (con minúscula)? ¿Un objeto? ¿Una clase? ¿O puede ser las ambas cosas al mismo tiempo?

R: El objeto "marca" está declarado donde dice "String marca nuevaInstancia String()".

"marca" ES un objeto y ES un atributo del tipo "String" que está en la clase "Marca". Pertenece o "es del tipo" String, como se indica en la declaración.

Es importante, en los ejemplos, diferencias las mayúsculas de las minúsculas: las clases normalmente comienzan con mayúscula y los atributos declaran objetos que empiezan con minúscula, por lo que podría tener, por ejemplo:

String string nuevaInstancia String()

Donde:

String: es la clase

string: es el objeto

14) P: ¿"Color colorDelAuto nuevinstancia Color()" es un objeto por estar dentro de la clase "Auto"?

R: Es un atributo, donde "colorDelAuto" es el objeto. Ese atributo está declarado en la clase "Auto".

15) P: ¿Pueden existir clases dentro de una clase que a su vez son objetos de esa clase en donde se declaran?

R: Si bien algunos lenguajes permiten anidar clases (declarar clases dentro de clases), no se considera una técnica de programación recomendada. A fines de los ejemplos de este curso, vamos a omitir esa práctica.

16) P: ¿Esto sería un getter?

metodo Float ingresarTemperatura()

mostrar: "Ingrese la temperatura en grados Kelvin"

ingresar: gradosKelvin

retornar: gradosKelvin

fin metodo

R: No, sería una mezcla de getter (porque devuelve el valor de un atributo) y de setter (porque permite ingresar un valor a un atributo).

¿Está mal? No. ¿Es un problema hacer algo así? No, en lo absoluto, simplemente que no es un getter, es un método cualquiera ("método de negocio"). ¿Conviene tener esta clase de método? Se recomienda que un método haga la menor cantidad de acciones posible, por lo que la alternativa sugerida sería tener:

metodo ingresarTemperatura()

mostrar: "Ingrese la temperatura en grados Kelvin"

ingresar: gradosKelvin

fin metodo

metodo Float devolverTemperatura()

retornar: gradosKelvin

fin metodo

De esta forma, habría dos invocaciones en lugar de una, pero los métodos sería más "atómicos" y más reutilizables.

17) P: ¿Qué relación hay entre "Color", "colorDelAuto", "primerAutoColor"? ¿Son distintos objetos de una misma clase y no tienen nada que ver uno con otro?

R: En la clase "EjemploAuto", los objetos "colorDelAuto" y "primerAutoColor" son instancias de la clase "Color".

18) P: ¿Qué son los métodos de negocio?

R: Un método de negocio es cualquier método que no sea el constructor. Un getter, un setter, un método para hacer un cálculo, un ordenamiento, una búsqueda, una validación, etc. Todo eso son métodos de negocio, para diferenciarlos de los constructores.

Lo que vimos

- Los primeros conceptos del paradigma de Programación Orientada a Objetos (POO)
- Los conceptos de clase, objeto, métodos y atributos
- Los conceptos de los dos primeros principios que hacen a la Orientación a Objetos (OO): abstracción y encapsulamiento

Lo que viene:

- Concepto de modificadores de acceso para clases, métodos y atributos
- Concepto de Herencia, uno los pilares de la POO
- Concepto de sobrecarga, tanto de operadores como de métodos