CURSO DE PROGRAMACIÓN FULL STACK

PROGRAMACIÓN ORIENTADA A OBJETOS

PARADIGMA ORIENTADO A OBJETOS

Objetivos de la Guía

En esta guía aprenderemos a:

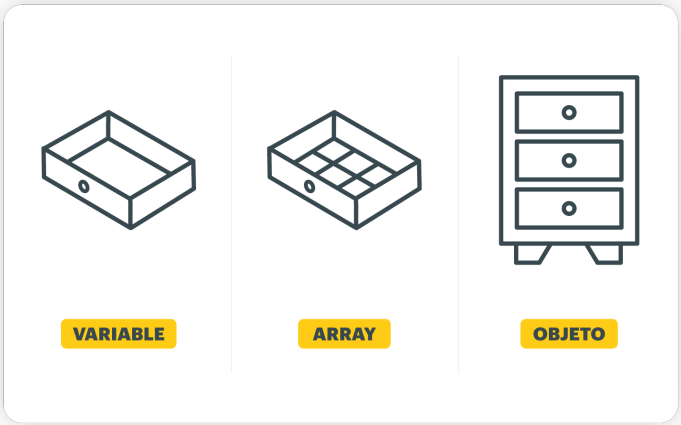
- Pensar y manejar objetos como una forma integrada de guardar información.
- Crear entidades en proyectos ordenados.
- Crear y distinguir constructores.
- Crear y distinguir getter & setter, sus usos y fundamentos.
- Crear clases de servicios con métodos que controlen los objetos.
- Utilizar entidades y clases de servicio desde el Main

¿QUE ES UN PARADIGMA DE PROGRAMACIÓN?

Un paradigma de programación es una manera o estilo de programación. Existen diferentes formas de diseñar un programa y varios modos de trabajar para obtener los resultados que necesitan los programadores. Por lo que un paradigma de programación se trata de un conjunto de métodos sistemáticos aplicables en todos los niveles del diseño de programas para resolver problemas.

¿QUÉ ES LA PROGRAMACIÓN ORIENTADA A OBJETOS?

Antes de sumergirnos en el concepto hagamos una revisión de cómo evolucionó nuestra manera de pensar y alojar información en la programación. En un primer momento, aprendimos las variables. Dijimos que las variables eran como cajones donde podíamos guardar sólo un dato de cierto tipo. Luego estudiamos los arrays, que nos permitían alojar varios datos a la vez, como si fuera un cajón con compartimentos; siempre y cuando estos datos fueran del mismo tipo. La programación orientada a objetos nos trae una nueva forma de almacenar información: OBJETOS. Pensaremos los objetos como un fichero o cajonera. Iremos profundizando este ejemplo a medida que avancemos en esta guía.



La **Programación Orientada a Objetos (POO)** es un paradigma de programación, es decir, un modelo o un estilo de programación que se basa en el concepto de **clases y objetos**. Este tipo de programación se utiliza para estructurar un programa de software en piezas simples y reutilizables de código (clases) para crear instancias individuales de objetos.

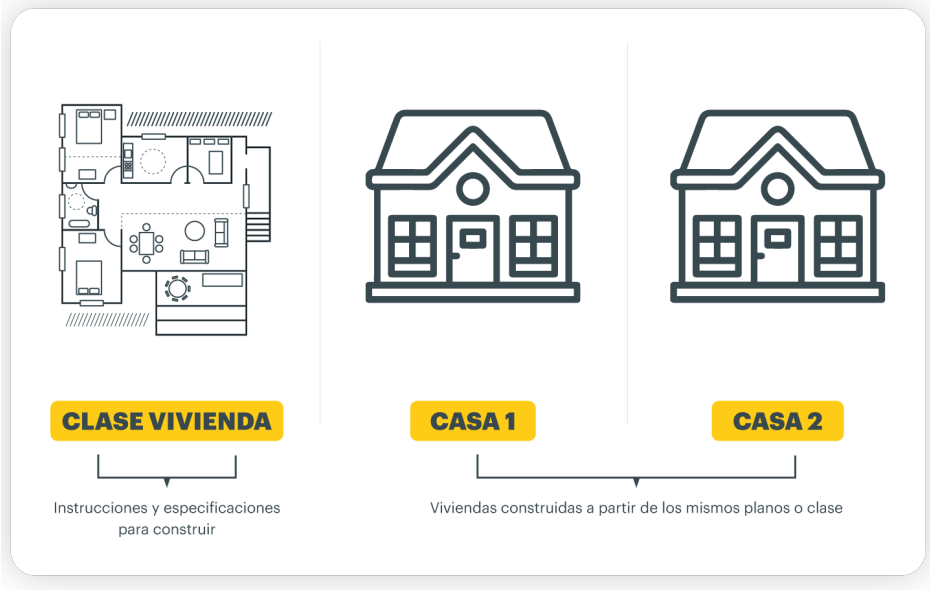
Con el paradigma de Programación Orientado a Objetos lo que buscamos es dejar de centrarnos en la lógica pura de los programas, para empezar a pensar en objetos, lo que constituye la base de este paradigma.

La programación orientada a objetos se enfoca en los **objetos, sus atributos y las interacciones** que se producen entre ellos para diseñar programas.

Un programa orientado a objetos es, esencialmente, un **conjunto de objetos** que se crean **interaccione entre sí** y dejan de existir cuando ya no son útiles durante la ejecución de un programa. Un programa puede llegar a ser muy complejo. La complejidad es más manejable cuando se descompone y se organiza en partes pequeñas y simples, los objetos.

¿POR QUÉ POO?

La Programación Orientada a objetos permite que el código sea reutilizable, organizado y fácil de mantener. Sigue el principio de desarrollo de *software* utilizado por muchos programadores **DRY *(Don't Repeat Yourself)***, para evitar duplicar el código y crear de esta manera programas eficientes. Además, evita el acceso no deseado a los datos o la exposición de código propietario mediante la encapsulación y la abstracción, de la que hablaremos en detalle más adelante.



PERO ENTONCES, ¿QUÉ ES UN OBJETO?

Un objeto en la programación es la forma que tenemos de llevar a código, la representación más fidedigna posible de un objeto de la vida real. Un objeto tendrá características (atributos) que son comunes a todos ellos, pero pueden variar entre sí.

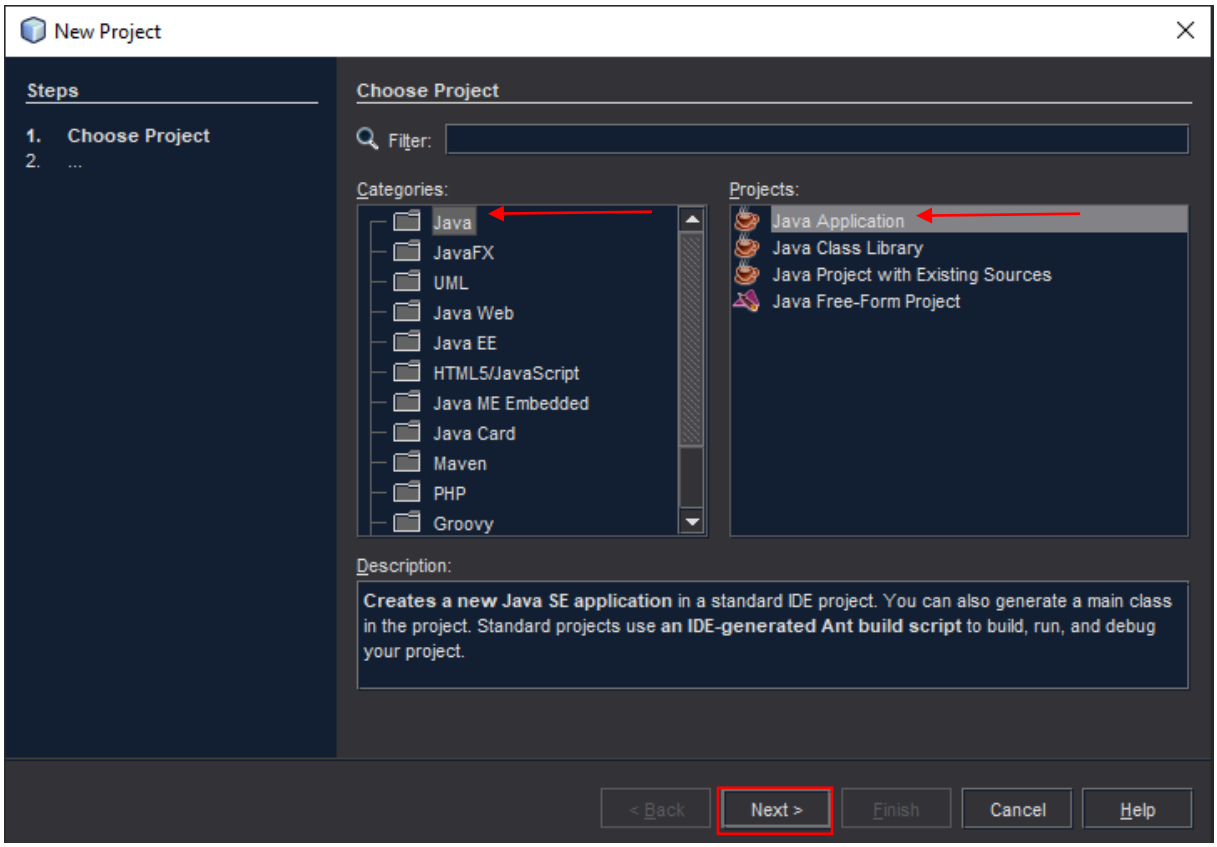
Volvamos al ejemplo del fichero o cajonera. El objeto es la cajonera, cada cajón representará un atributo. Una vez creado el molde para crear este objeto cajonera, cada uno de sus cajones alojará un tipo de dato que será el mismo tipo pero la información que se aloje dentro será diferente. En el primer cajón debemos guardar remeras, pero estas pueden ser rojas en una cajonera o verdes en otra.

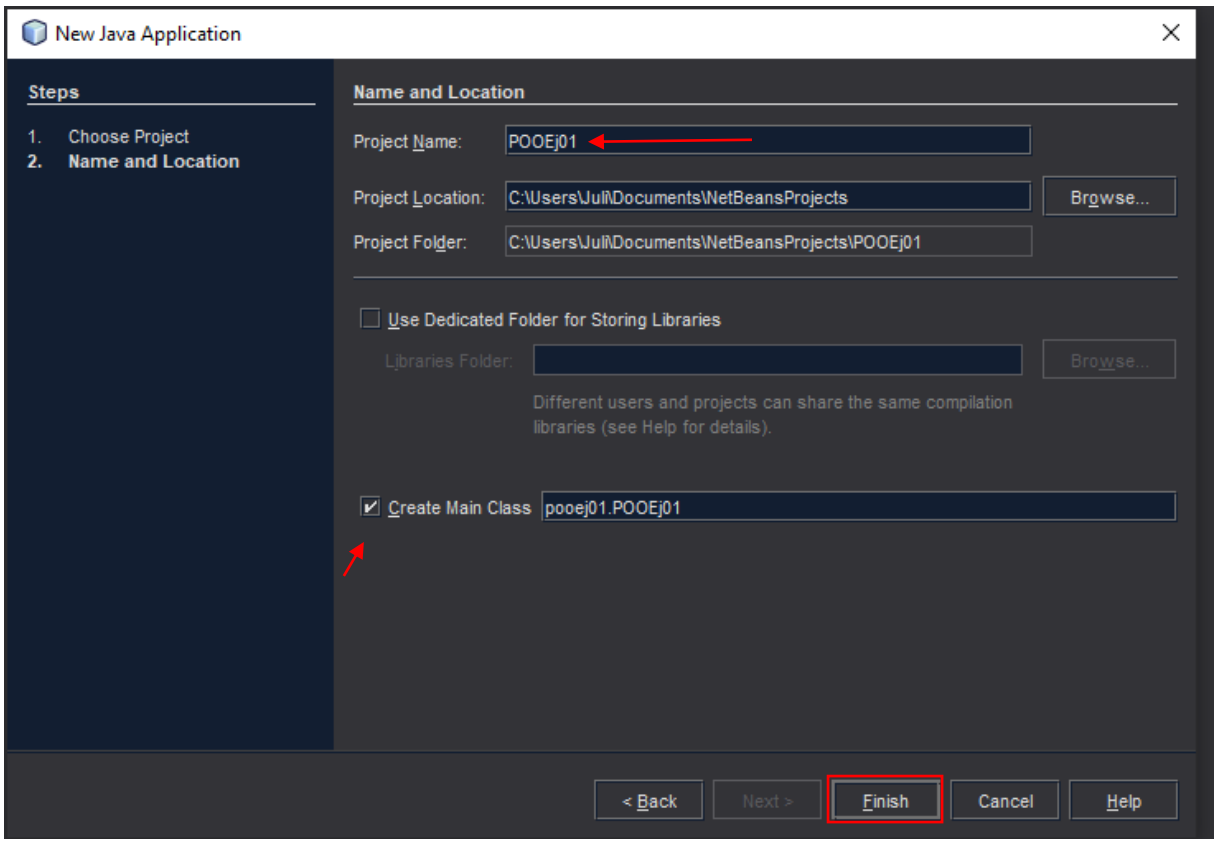
Pensemos también a las personas como objetos. Tenemos ciertas características que son uniformes a todos. Todos tenemos un nombre, un número de identificación, una fecha de nacimiento. Estas características variarían de una persona a otra, pero al compartir esta información podemos elaborar código que represente y nos permita identificar a una o varias personas.

¿CÓMO CREAMOS OBJETOS?

Para llevar cierto orden, por cada uno de los ejercicios de esta guía **crearemos un nuevo proyecto** en NetBeans.

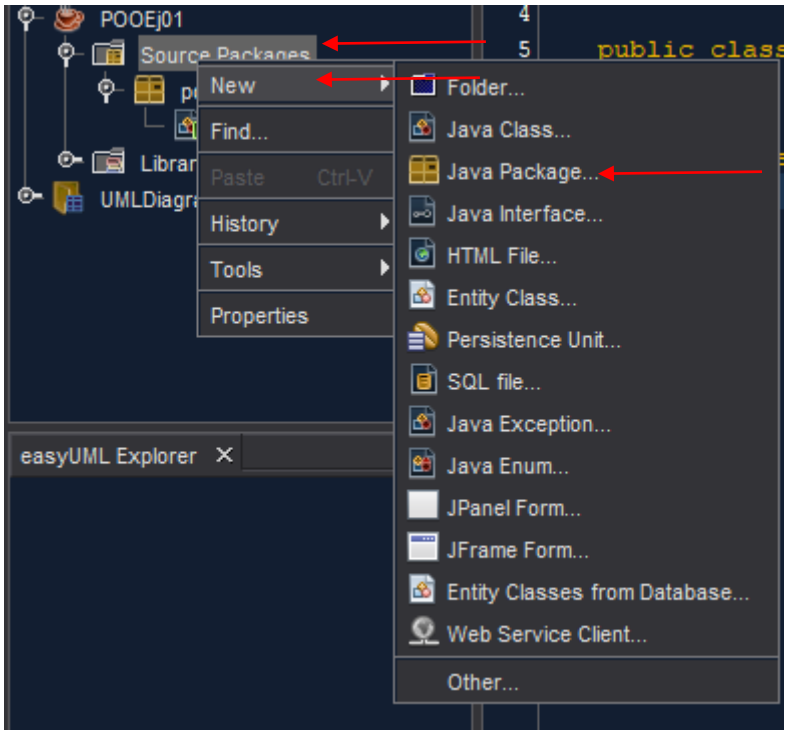
Primero crearemos el proyecto:

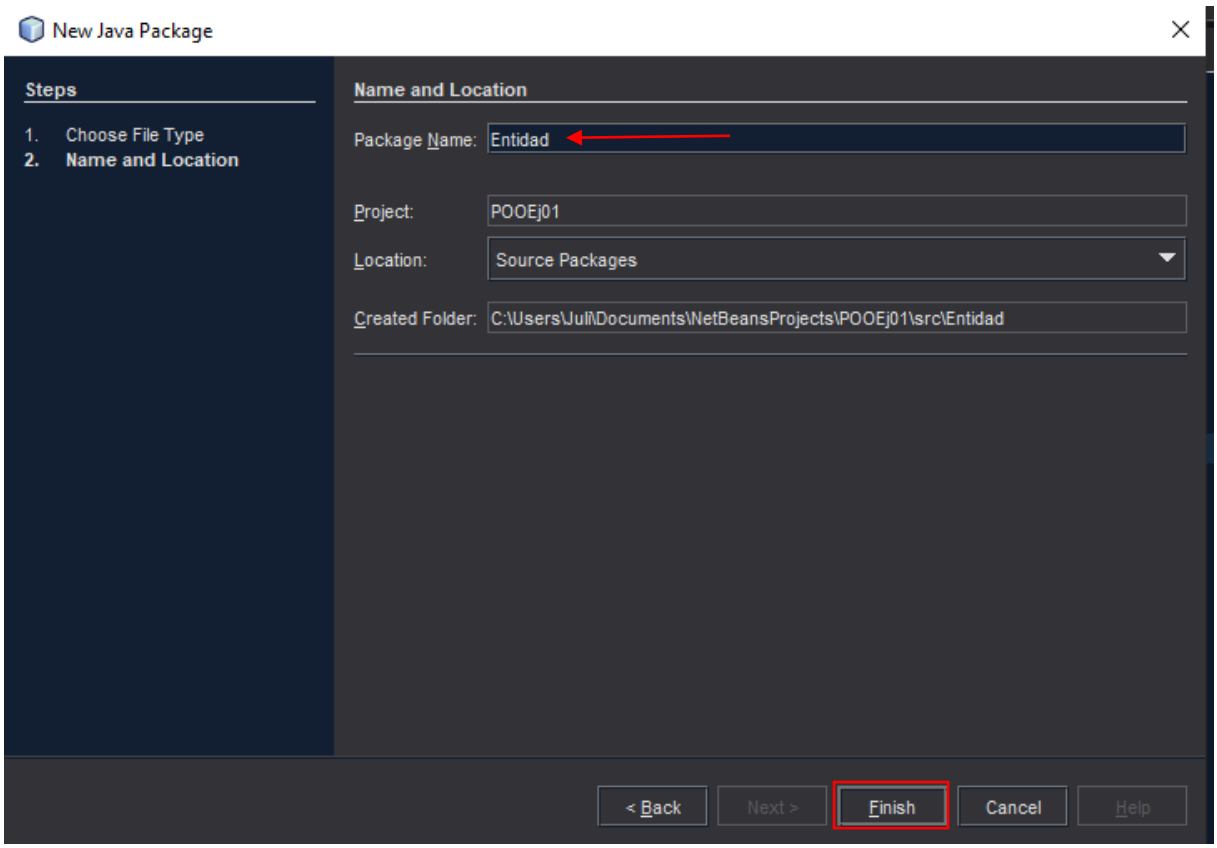


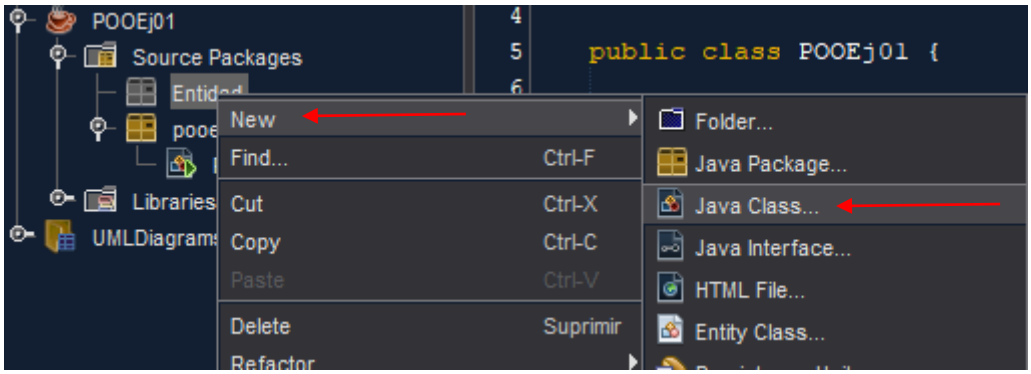


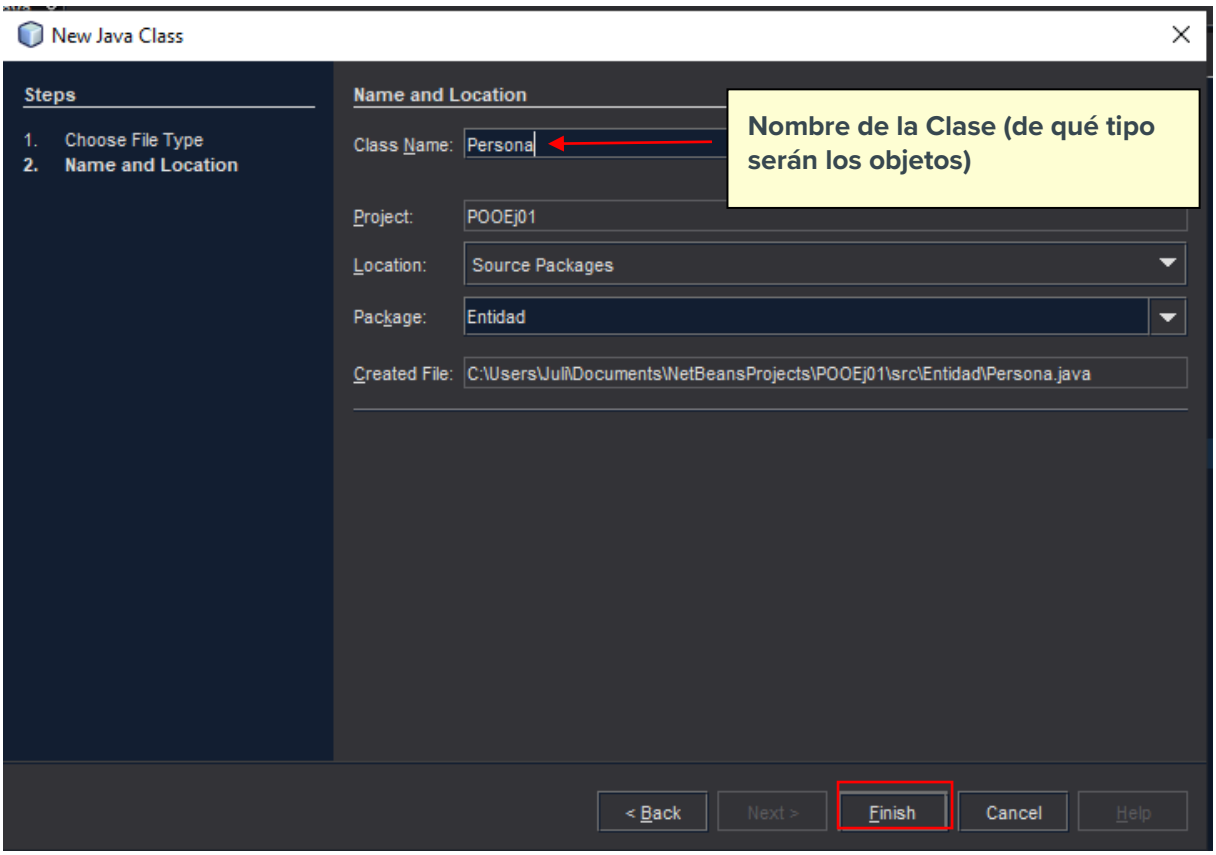
Luego, necesitamos crear una clase. Una **clase es un molde** para crear múltiples objetos que encapsula datos y comportamiento. Una clase es una combinación específica de atributos y métodos y puede considerarse un tipo de dato de cualquier tipo no primitivo.

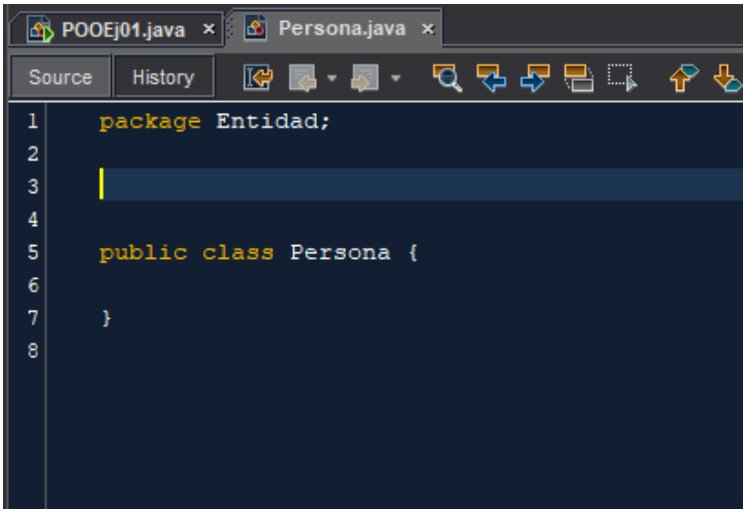
Las clases deberán guardarse en un paquete llamado Entidad:




New Java Package
Steps
1. Choose File Type
2. Name and Location
Name and Location
Package Name: Entidad
Project: POOEj01
Location: Source Packages
Created Folder: C:\Users\Juli\Documents\NetBeansProjects\POOEj01\src\Entidad
< Back
Next >
Finish
Cancel
Help


POOEj01
Source Packages
Libraries
UMLDiagrams
4
5
public class POOEj01 {
New
Find... Ctrl-F
Cut Ctrl-X
Copy Ctrl-C
Paste Ctrl-V
Delete Suprimir
Refactor
Folder...
Java Package...
Java Class...
Java Interface...
HTML File...
Entity Class...





Así, una clase es una especie de plantilla o prototipo de objetos: define los atributos que componen ese tipo de objetos y los métodos que pueden emplearse para trabajar con esos objetos. En su forma más simple, una clase se define por la palabra reservada class seguida del nombre de la clase. El nombre de la clase debe empezar por mayúscula.

Si el nombre es compuesto, entonces cada palabra debe empezar por mayúscula. La definición de la clase se pone entre las llaves de apertura y cierre.

```
public class NombreClase {

	// atributos
	// constructores
	// metodos propios

}
```

MANOS A LA OBRA!

EJERCICIO PERSONA

Es tu turno, crea tu propio proyecto con la clase Persona.

ESTADO Y COMPORTAMIENTO

En términos más generales, **un objeto es una abstracción conceptual del mundo real** que se puede **traducir a un lenguaje de programación** orientado a objetos. Los objetos del mundo real comparten dos características: Todos poseen **estado y comportamiento**. Por ejemplo, el perro tiene estado (color, nombre, raza, edad) y el comportamiento (ladrar, caminar, comer, acostarse, mover la cola). Por lo tanto, un estado permite informar cuáles son las características del objeto y lo que este representa, y el comportamiento, consiste en decir lo que sabe hacer.

El estado de un objeto es una **lista de variables conocidas como sus atributos**, cuyos valores representan el estado que caracteriza al objeto.

El comportamiento es una **lista de métodos, procedimientos, funciones u operaciones** que un objeto puede ejecutar a solicitud de otros objetos. Los objetos también se conocen como instancias.

ELEMENTOS DE UNA CLASE

Una clase describe un tipo de objetos con características comunes. Es necesario definir la información que almacena el objeto y su comportamiento.

¿QUÉ SON LOS ATRIBUTOS?

Los atributos son **características comunes** a todos los objetos. Son los “espacios” donde **alojaremos información** que cambiará en cada objeto pero que corresponda a una descripción del mismo. Son los “cajones” de la cajonera.

El estado o información de un objeto se almacena en atributos. Los atributos pueden ser de tipos primitivos de Java (descriptos en la guía Intro Java) o del tipo de otros objetos. La declaración de un atributo de un objeto tiene la siguiente forma:

```
<modificador>* <tipo> <nombre>
```

- **<modificador>:** si bien hay varios valores posibles para el <modificador>, por el momento solo usaremos modificadores de visibilidad: public, protected, private.
- **<tipo>:** indica la clase a la que pertenece el atributo definido.
- **<nombre>:** puede ser cualquier identificador válido y denomina el atributo que está siendo declarado.

¿NECESITAS UN **EJEMPLO**?

```
5   public class Persona {
6
7       public String nombre;
8
9   }
10
```

Estos atributos irán al principio de la clase.

MANOS A LA OBRA!

EJERCICIO PERSONA – ATRIBUTOS

Seguiremos trabajando sobre la clase Persona que creamos y ahora deberás sumarle 3 atributos que creas pertinentes.

¿QUÉ SON LOS CONSTRUCTORES?

Los constructores con métodos propios del objeto que nos permiten CREARLO. A la creación de un objeto se le denomina INSTANCIACIÓN.

Además de definir los atributos de un objeto, es necesario definir los métodos que determinan su comportamiento. Toda clase debe definir un método especial denominado constructor para instanciar los objetos de la clase. Este método tiene el mismo nombre de la clase. La declaración básica toma la siguiente forma:

¿NECESITAS UN **EJEMPLO**?

Este es un constructor por defecto

```
9       public Persona() {
10      }
```

Éste es un constructor con parámetros

```
12      public Persona(String nombre) {
13          this.nombre = nombre;
14      }
15
```

```
[<modificador>] <nombre de clase> ( <argumento>* ) {
   <sentencia>*
}
```

- **<nombre de clase>:** El nombre del constructor debe ser siempre el mismo que el de la clase.
- **<modificador>:** Actualmente, los únicos modificadores válidos para los constructores son public, protected y private.
- <**argumentos>:** es una lista de parámetros que tiene la misma función que en los métodos.

El método constructor se ejecuta cada vez que se instancia un objeto de la clase. Este método se utiliza para **inicializar** los atributos del objeto que se instancia.

Para diferenciar entre los atributos del objeto y los identificadores de los parámetros del método constructor, se utiliza la palabra this. De esta forma, los parámetros del método pueden tener el mismo nombre que los atributos de la clase.

La instanciación de un objeto consiste en asignar un espacio de memoria al que se hace referencia con el nombre del objeto. Los identificadores de los objetos permiten acceder a los valores almacenados en cada objeto.

Al hacer click derecho sobre la clase se despliega un menú de opciones que incluye una que dice “Insert code”, inmediatamente se despliega otra que dice “Generate”, debemos seleccionar la que dice “Constructor”. Al abrirse la ventana emergente, podemos no seleccionar ningún atributo, lo que nos creará un constructor por defecto o todos, creando así el constructor por parámetro.

El constructor por defecto

Cada clase tiene al menos un constructor. Si no se escribe un constructor, el lenguaje de programación Java le provee uno por defecto. Este constructor no posee argumentos y tiene un cuerpo vacío. Si se define un constructor que no sea vacío, el constructor por defecto se pierde, salvo que creemos un nuevo constructor vacío.

Una vez que se ha declarado una clase, se pueden crear objetos a partir de ella. A la creación de un objeto se le denomina **instanciación**. Por esta razón que se dice que un objeto es una instancia de una clase y el término instancia y objeto se utilizan indistintamente. Para crear objetos, basta con declarar una variable de alguno de los tipos de las clases definidas.

```
NombreClase nombreObjeto;
```

Para crear el objeto y asignar un espacio de memoria es necesario realizar la instanciación con el operador new. El operador new instancia el objeto y reserva espacio en memoria para los atributos y devuelve una referencia que se guarda en la variable.

```
nombreObjeto = new nombreClase();
```

Tanto la declaración de un objeto como la asignación del espacio de memoria se pueden realizar en un solo paso:

```
NombreClase nombreObjeto = new NombreClase();
```

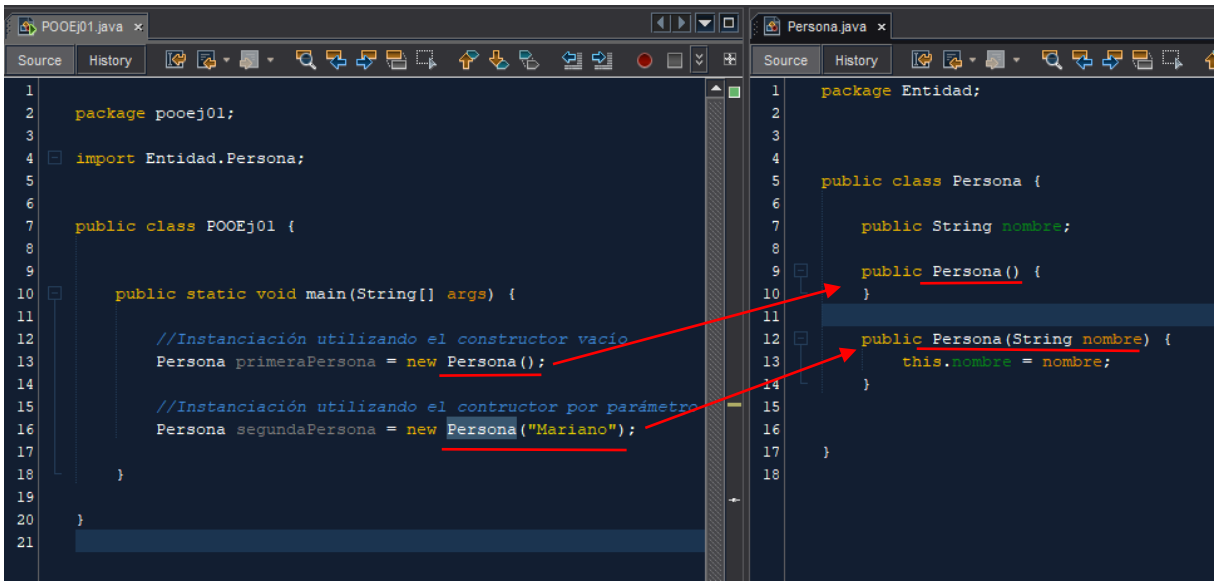
A partir de este momento los objetos ya pueden ser referenciados por su nombre.

Al instanciar objetos desde el main u otra clase, verás que te aparece una señal de alerta en el contador de líneas. En ella te pedirá que importes la clase que deseas utilizar para acceder a su información.

¿NECESITAS UN **EJEMPLO**?

```java
package pooej01;

import Entidad.Persona;

public class POOEj01 {

    public static void main(String[] args) {

        //Instanciación utilizando el constructor vacío
        Persona primeraPersona = new Persona();

        //Instanciación utilizando el contructor por parámetro
        Persona segundaPersona = new Persona("Mariano");

    }

}
```

Instanciamos dos objetos en el main del tipo Persona



¿Cómo se crean los constructores?

Dentro de la clase, para agregar automáticamente los constructores podemos presionar el botón derecho > Insert Code > Constructor y podemos no seleccionar ningún atributo o seleccionar todos los atributos.

MANOS A LA OBRA!

EJERCICIO PERSONA – CONSTRUCTORES

Volveremos a usar la clase Persona y vamos a crear objetos en tu Main utilizando los constructores.

¿Te diste cuenta de que instanciar un objeto es llamar al método constructor?

ABSTRACCIÓN Y ENCAPSULAMIENTO

La abstracción es la habilidad de ignorar los detalles de las partes para enfocar la atención en un nivel más alto de un problema. El encapsulamiento sucede cuando algo es envuelto en una capa protectora. Cuando el encapsulamiento se aplica a los objetos, significa que los datos del objeto están protegidos, "ocultos" dentro del objeto. Con los datos ocultos, ¿cómo puede el resto del programa acceder a ellos? (El acceso a los datos de un objeto se refiere a leerlos o modificarlos.) El resto del programa no puede acceder de manera directa a los datos de un objeto; lo tiene que hacer con ayuda de los métodos del objeto. Al hecho de proteger los datos o atributos con los métodos se denomina encapsulamiento.

ABSTRACCIÓN

La abstracción es la propiedad que considera los aspectos más significativos o notables de un problema y expresa una solución en esos términos. La abstracción posee diversos grados o niveles de abstracción, los cuales ayudan a estructurar la complejidad intrínseca que poseen los sistemas del mundo real. La abstracción encarada desde el punto de vista de la programación orientada a objetos es el mecanismo por el cual se proveen los límites conceptuales de los objetos y se expresan sus características esenciales, dejando de lado sus características no esenciales. Si un objeto tiene más características de las necesarias los mismos resultan difíciles de usar, modificar, construir y comprender. En el análisis hay que concentrarse en ¿Qué hace? y no en ¿Cómo lo hace?

ENCAPSULAMIENTO

La encapsulación o encapsulamiento significa reunir en una cierta estructura a todos los elementos que a un cierto nivel de abstracción se pueden considerar pertenecientes a una misma entidad, y es el proceso de agrupamiento de datos y operaciones relacionadas bajo una misma unidad de programación, lo que permite aumentar la cohesión de los componentes del sistema.

El encapsulamiento **oculta lo que hace un objeto** de lo que hacen otros objetos y del mundo exterior por lo que se denomina también ocultación de datos. Un objeto tiene que presentar "una cara" al mundo exterior de modo que se puedan iniciar sus operaciones.

Los métodos operan sobre el estado interno de un objeto y sirven como el mecanismo primario de comunicación entre objetos. Ocultar el estado interno y hacer que toda interacción sea a través de los métodos del objeto es un mecanismo conocido como encapsulación de datos.

MODIFICADORES DE ACCESO

Para lograr el uso correcto del encapsulamiento vamos a utilizar los modificadores de acceso, estos, van a dejarnos elegir como se accede a los datos y a través de que se accede a dichos datos. Todas las clases poseen diferentes niveles de acceso en función del modificador de acceso (visibilidad). A continuación, se detallan los niveles de acceso con sus símbolos correspondientes:

- **Public:** Este modificador permite a **acceder a los elementos desde cualquier clase**, independientemente de que esta pertenezca o no al paquete en que se encuentra el elemento.

- **Private:** Es el modificador más restrictivo y especifica que los elementos que lo utilizan **sólo pueden ser accedidos desde la clase en la que se encuentran**. Este modificador sólo puede utilizarse sobre los atributos de una clase y sobre interfaces y clases internas, no sobre clases o interfaces de primer nivel, dado que esto no tendría sentido. Es importante destacar también que el modificador private convierte los elementos en privados para otras clases, no para otras instancias de la clase. Es decir, un objeto de una determinada clase puede acceder a los atributos privados de otro objeto de la misma clase.

- **Protected:** Este modificador indica que los elementos **sólo pueden ser accedidos desde su mismo paquete y desde cualquier clase que extienda la clase** en que se encuentra, independientemente de si esta se encuentra en el mismo paquete o no. Este modificador, como private, no tiene sentido a nivel de clases o interfaces no internas.

Si no especificamos ningún modificador de acceso se utiliza el nivel de acceso por defecto (Default), que consiste en que el elemento puede ser accedido sólo desde las clases que pertenezcan al mismo paquete. Los distintos modificadores de acceso quedan resumidos en la siguiente tabla:

Visibilidad	Public	Private	Protected	Default
Desde la misma Clase	SI	SI	SI	SI
Desde cualquier Clase del mismo Paquete	SI	NO	SI	SI
Desde una Subclase del mismo Paquete	SI	NO	SI	SI
Desde una Subclase fuera del mismo Paquete	SI	NO	SI, a través de la herencia	NO
Desde cualquier Clase fuera del Paquete	SI	NO	NO	NO

MÉTODOS PROPIOS

Los métodos son funciones que determinan el comportamiento de los objetos. Un objeto se comporta de una u otra forma dependiendo de los métodos de la clase a la que pertenece. Todos los objetos de una misma clase tienen los mismos métodos y el mismo comportamiento. Para definir los métodos, el lenguaje de programación Java toma la siguiente forma básica:

```
<modificador>* <tipo de retorno> <nombre> ( <argumento>>* ) {

    <sentencias>*

    return valorRetorno;

}
```

- **<modificador>:** el segmento es opcional y puede contener varios modificadores diferentes incluyendo a public, protected y private. Aunque no está limitado a estos.

- **<tipo de retorno>:** el tipo de retorno indica el tipo de valor devuelto por el método. Si el método no devuelve un valor, debe ser declarado void. La tecnología Java es rigurosa acerca de los valores de retorno. Si el tipo de retorno en la declaración del método es un int, por ejemplo, el método debe devolver un valor int desde todos los posibles caminos de retorno (y puede ser invocado solamente en contextos que esperan un int para ser devuelto). Se usa la sentencia return dentro de un método para devolver un valor.

- **<nombre>:** puede ser cualquier identificador válido, con algunas restricciones basadas en los nombres que ya están en uso.

- **<argumento>:** permite que los valores de los argumentos sean pasados hacia el método. Los elementos de la lista están separados por comas y cada elemento consiste en un tipo y un identificador.

Existen tres tipos de métodos: métodos de consulta, métodos modificadores y operaciones. Los métodos de consulta sirven para extraer información de los objetos, los métodos modificadores sirven para modificar el valor de los atributos del objeto y las operaciones definen el comportamiento de un objeto.

GETTER & SETTER

Para acceder a los atributos de un objeto se definen los **métodos get y set**. Los métodos get se utilizan para **consultar el estado de un objeto** y los métodos set para **modificar su estado**. Un método get se declara public y a continuación se indica el tipo de dato que devuelve. Es un método de consulta. La lista de parámetros de un método get queda vacía. En el cuerpo del método se utiliza return para devolver el valor correspondiente al atributo que se quiere devolver, y al cual se hace referencia como **this.nombreAtributo.**

Por otra parte, un método set se declara public y devuelve void. La lista de parámetros de un método set incluye el tipo y el valor a modificar. Es un método modificador. El cuerpo de un método set asigna al atributo del objeto el parámetro de la declaración.

¿NECESITAS UN **EJEMPLO**?

```
16      public String getNombre() {
17          return nombre;
18      }
19
20      public void setNombre(String nombre) {
21          this.nombre = nombre;
22      }
23
24
```

¿Cómo se crean los getter y setter?

Dentro de la clase, para agregar automáticamente los getter & setter podemos presionar el botón derecho > Insert Code > Getter & Setter y selecciono todos los atributos.

¿CÓMO UTILIZAMOS ESTOS MÉTODOS?

Un método se puede invocar dentro o fuera de la clase donde se ha declarado. Si el método se invoca dentro de la clase, basta con indicar su nombre**. Si el método se invoca fuera de la clase** entonces se debe indicar el nombre del objeto y el nombre del método. Cuando se invoca a un método ocurre lo siguiente:

- En la línea de código del programa donde se invoca al método se calculan los valores de los argumentos.
- Los parámetros se inicializan con los valores de los argumentos.
- Se ejecuta el bloque código del método hasta que se alcanza return o se llega al final del bloque.
- Si el método devuelve un valor, se sustituye la invocación por el valor devuelto.
- La ejecución del programa continúa en la siguiente instrucción donde se invocó el método.

¿NECESITAS UN **EJEMPLO**?

```
18          primeraPersona.setNombre("Lucio");
19
20          segundaPersona.getNombre();
21
```

En nuestro ejemplo de “cajonera” el método Get es ir y tomar una fotografía de lo que guardamos en el cajón. El método Set es tomar una nueva información y guardarla en el cajón. Si ya estaba ocupado, se quita el contenido anterior y se reemplaza por el nuevo.

MANOS A LA OBRA!

EJERCICIO PERSONA – GET Y SET

Continuaremos con nuestra clase Persona y vamos a primero crear los getter y setter y después invocarlos desde el Main con alguno de los objetos que instanciaste.

¿Todos los atributos deben tener Getters&Setters?

No necesariamente, si tuvieramos por ejemplo los atributos fechaDeNacimiento y edad, podemos utilizar la fecha de nacimiento para calcular la edad y no permitir que la edad se setee manualmente. Pondremos sólo un get para acceder a ella en caso de que sea necesario.

ATRIBUTOS Y MÉTODOS ESTÁTICOS

Un atributo o un método de una clase se puede modificar con la palabra reservada static para indicar que este atributo o método no pertenece a las instancias de la clase si no a la propia clase.

Se dice que son atributos de clase si se usa la palabra clave static: en ese caso la variable es única para todas las instancias (objetos) de la clase (ocupa un único lugar en memoria), es decir que, si se poseen múltiples instancias de una clase, cada una de ellas no tendrán una copia propia de este atributo, si no que todas estas instancias compartirán una misma copia del atributo. A veces a las variables de clase se les llama variables estáticas. Si no se usa static, el sistema crea un lugar nuevo para esa variable con cada instancia (la variable es diferente para cada objeto).

En el caso de una constante no tiene sentido crear un nuevo lugar de memoria por cada objeto de una clase que se cree. Por ello es adecuado el uso de la palabra clave static. Cuando usamos ”static final” se dice que creamos una constante de clase, un atributo común a todos los objetos de esa clase.

```
public class Cuenta {

private static String saldo;

}
```

ATRIBUTOS FINALES

En este contexto indica que una variable es de tipo constante: no admitirá cambios después de su declaración y asignación de valor. La palabra reservada **final** determina que un atributo no puede ser sobrescrito o redefinido, es decir, no funcionará como una variable “tradicional”, sino como una constante. Toda constante declarada con *final* ha de ser inicializada en el mismo momento de declararla. El modificador *final* también se usa como palabra clave en otro contexto: una clase *final* es aquella que no puede tener clases que la hereden. Lo veremos más adelante cuando hablemos sobre herencia.

Cuando se declaran constantes es muy frecuente que los programadores usen letras mayúsculas (como práctica habitual que permite una mayor claridad en el código), aunque no es obligatorio.

```
Public class Perro {

private final int edad;

}
```

CLASE SERVICIO

La clase servicio (service) o control, va a ser una clase auxiliar que nos va a ayudar con el manejo de las clases y los objetos de esas clases, pero para poder explicar esto, primero vamos a tener que ver **los patrones generales de software GRASP**. Aunque se considera que más que **patrones** propiamente dichos, son una serie de "buenas prácticas" de aplicación recomendable en el diseño de software.

PATRONES GRASP

Un proceso de desarrollo sirve para normalizar quien hace que cosa en cada momento y como debe realizarse esta cosa.

GRASP es el acrónimo de General **Responsibility Assignment Software Patterns**. Una de las cosas más complicadas en Orientación a Objeto consiste en elegir las clases adecuadas decidir como estas clases deben interactuar.

PATRON EXPERTO

Dentro de los patrones GRASP, vamos a utilizar el patrón experto. El GRASP de experto en información es el principio básico de asignación de responsabilidades. Nos indica, por ejemplo, que la responsabilidad de la creación de un objeto o la implementación de un método debe recaer sobre la clase que conoce toda la información necesaria para crearlo. De este modo obtendremos un diseño con mayor cohesión y así la información se mantiene encapsulada (disminución del acoplamiento).

Problema

¿Cuál es el principio general para asignar responsabilidades a los objetos?

Solución

Asignar una responsabilidad al experto en información.

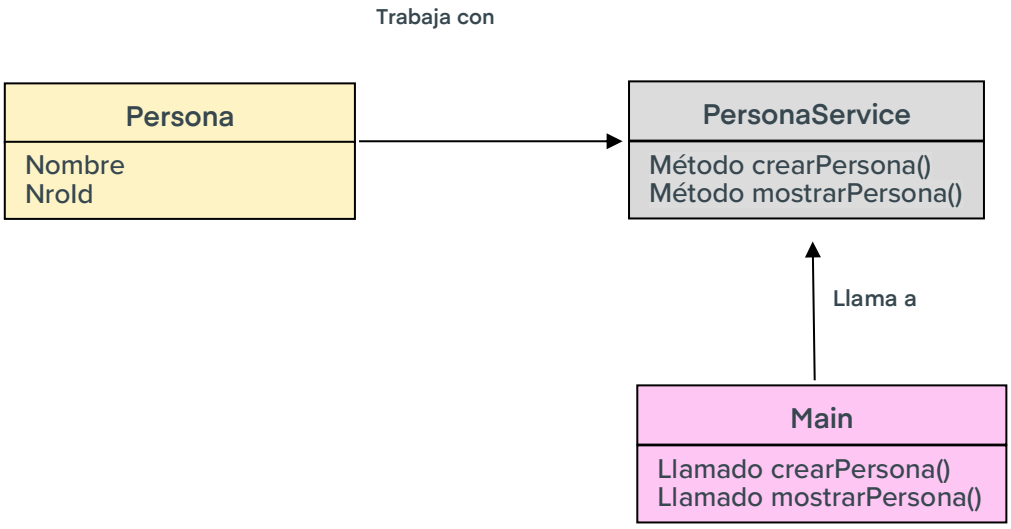
Beneficios

Se mantiene el encapsulamiento, los objetos utilizan su propia información para llevar a cabo sus tareas. Se distribuye el comportamiento entre las clases que contienen la información requerida. Son más fáciles de entender y mantener.

CLASE SERVICIO

Esta clase del patrón experto va a ser la clase servicio. Es una clase común y corriente pero que se va a encargar de crear los objetos y va a tener todos los métodos necesarios para la utilización de ese objeto. Supongamos que necesitamos un método que le sume un valor x a un atributo del objeto, ese método estará en la clase control.

Siempre se crea una clase control, por cada clase que tengamos, si tenemos las clases Persona y Sueldo, crearemos una clase control para Persona y otra para Sueldo. La idea es que una clase servicio, se encargue de solo una clase.



Como podemos ver en el diagrama, PersonaService llama y trabaja con la clase Persona y crea instancias de ese objeto a través de un método, también, tiene un método para mostrar la persona.

Pero, el encargado de llamar a esos métodos de la clase PersonaService va a ser la clase Main, que vendría a ser el usuario llamando a las cosas que quiere realizar con la Persona.

Para trabajar con clase de servicio o control debemos crear un nuevo paquete en nuestro proyecto que se llame Servicios.

METODOS CLASE SERVICIO

Como pudimos ver en el diagrama anterior la clase PersonaService, tenía sus propios métodos, que iban a trabajar con la clase Persona, pero no que no se encuentran en la clase Persona, sino en PersonaService. Es importante que a la hora de trabajar con métodos dentro de la clase servicio o métodos externos, tengamos en cuenta las siguientes cosas:

Parámetros y argumentos

Los parámetros de un método definen la cantidad y el tipo de dato de los valores que recibe un método para su ejecución. Los argumentos son los valores que se pasan a un método durante su invocación. El método recibe los argumentos correspondientes a los parámetros con los que ha sido declarado. Un método puede tener tantos parámetros como sea necesario. La lista de parámetros de la cabecera de un método se define con la siguiente sintaxis:

```
tipo nombre [tipo nombre,]
```

Durante la invocación de un método es necesario que el número y el tipo de argumentos coincidan con el número y el tipo de parámetros declarados en la cabecera del método. Durante el proceso de compilación se comprueba que durante la invocación de un método se pasan tantos argumentos como parámetros tiene declarados y que además coinciden los tipos. Esta es una característica de los lenguajes que se denominan “strongly tiped” o “fuertemente tipado”

Paso de parámetros

Cuando se invoca un método se hace una copia de los valores de los argumentos en los parámetros. Esto quiere decir que, si el método modifica el valor de un parámetro, nunca se modifica el valor original del argumento.

Cuando se pasa una referencia a un objeto se crea un nuevo alias sobre el objeto, de manera que esta nueva referencia utiliza el mismo espacio de memoria del objeto original y esto permite acceder al objeto original.

El valor de retorno

Un método puede devolver un valor. Los métodos que no devuelven un valor se declaran void, mientras que los métodos que devuelven un valor indican el tipo que devuelven: int, double, char, String o un tipo de objeto.

Sólo utiliza métodos void para mostrar información. Si realizarás cambios en los atributos del objeto, envía ese objeto por parámetro y retornalo con los cambios realizados

Sobrecarga de métodos

La sobrecarga de métodos es útil para que el mismo método opere con parámetros de distinto tipo o que un mismo método reciba una lista de parámetros diferente. Esto quiere decir que puede haber dos métodos con el mismo nombre que realicen dos funciones distintas. La diferencia entre los métodos sobrecargados está en su declaración, y más específicamente, en la cantidad y tipos de datos que reciben.

¿NECESITAS UN **EJEMPLO**?

```java
    public Persona crearPersona(){
        //Instanciamos un objeto persona con sus atributos vacios
        Persona personaCompleta = new Persona();

        //Pedimos al usuario que ingrese la informacion
        //que se alojará en el atributo por consola
        System.out.println("Ingrese el nombre de la persona");

        //Utilizamos el objeto para invocar al método SET
        //Y con el Scanner recibimos la información
        personaCompleta.setNombre(leer.next());


        //este método retorna un objeto persona son sus atributos
        //llenos de información
        return personaCompleta;
    }
```

En el Main:

```java
            //Debemos instanciar un objeto del tipo Servicio
            //para acceder a sus métodos
            PersonaService persServicio = new PersonaService();

            //Alojamos el retorno del método en un objeto tipo Persona
            Persona terceraPersona = persServicio.crearPersona();

        }
```

MANOS A LA OBRA!

EJERCICIO VOID

Crea un método void que reciba un objeto tipo persona como parámetro y utilice el get para mostrar sus atributos. Llama ese método desde el main.

EN RESUMEN

Antes de POO, la técnica estándar de programación era la programación procedural. Se denomina programación procedural porque en ella se destacan los procedimientos o tareas que resuelven un problema. Se piensa primero en lo que se quiere hacer: los procedimientos.

En contraste, el paradigma POO invita a pensar en lo que se desea que represente el programa. Normalmente se responde esta invitación identificando algunas cosas en el mundo que se desea que el programa modele.

Estas cosas podrían ser entidades físicas o conceptuales, por ejemplo, un libro. Una vez identificadas las cosas que se quiere modelar, se identifican sus propiedades/atributos básicos. Estos se pueden agrupar todos juntos en una estructura coherente llamada objeto que creamos a través de las clases.

Pueden encontrar un ejemplo para descargar de POO en el GIT de tu Curso.

El ejemplo está pensado para una página que va a administrar perros en su página web, muestra las distintas maneras de llenar y de mostrar un objeto.

CLASES DE UTILIDAD PARTE 2

Recordemos que las clases de utilidad son clases dentro del API de Java que son muy utilizadas en el desarrollo de aplicaciones. Las clases de utilidad son clases que definen un conjunto de métodos que realizan funciones, normalmente muy reutilizadas. Estas nos van a ayudar junto con las estructuras de control, a lograr resolver problemas de manera más sencilla.

Entre las clases de utilidad de Java más utilizadas y conocidas están las siguientes: Arrays, String, Integer, Math, Date, Calendar y GregorianCalendar. En la guía anterior vimos solo las clases Math y String. Ahora vamos a ver el resto de las clases.

CLASE ARRAYS

La clase Arrays es una clase de utilidad que posee una gran cantidad de métodos para manipular arreglos.

Método	Descripción
Arrays.equals(arreglo1, arreglo2)	Retorna true o false, si dos arreglos del mismo tipo de dato son iguales.
Arrays.fill(arreglo, variable) **Arrays.fill(arreglo, int desde, int hasta, variable)**	Este método lo que hace es inicializar todo el arreglo con la variable o valor que pasamos como argumento. Este método se puede usar con cualquier tipo de dato y le podemos decir desde y hasta que índice queremos que llene con ese valor.
Arrays.sort(arreglo) **Arrays.sort(arreglo, int desde, int hasta)**	Este método sirve para ordenar un arreglo de manera ascendente. A este método también le podemos decir desde y hasta que índice queremos que ordene.
Arrays.toString(arreglo)	Este método imprime el arreglo como una cadena, la cadena consiste en una lista de los elementos del arreglo encerrados entre corchetes ("[]"). Los elementos adyacentes están separados por comas (",").

CLASE INTEGER

La clase Integer permite convertir un tipo primitivo de dato int a objeto Integer. La clase Integer pertenece al paquete java.lang del API de Java y hereda de la clase java.lang.Number.

Método	Descripción
Integer(String s)	Constructor que inicializa un objeto con una cadena de caracteres. Esta cadena debe contener un número entero.
compareTo(entero, otroEntero)	Compara dos objetos Integer numéricamente. Retorna 0 si son iguales, entero negativo si el primer número es menor o entero positivo si el primer número es mayor.
doubleValue()	Retorna el valor del Integer en tipo primitivo double
equals(Object obj)	Compara el Integer con el objeto del parámetro. Devuelve true si son iguales y false si no.
parseInt(String s)	Convierte la cadena de caracteres numérica del parámetro en tipo primitivo int.
toString()	Retorna el valor del Integer en una cadena de caracteres.

CLASE DATE

La clase Date modela objetos o variables de tipo fecha. La clase Date representa un instante de tiempo específico con una precisión en milisegundos y permite el uso del formato Universal Coordinated Time (UTC). Por otro lado, muchas computadoras están definidas en términos de Greenwich Mean Time (GMT) que es equivalente a Universal Time (UT). GMT es el nombre estándar y UT es el nombre científico del estándar. La diferencia entre UT y UTC es que UTC está basado en un reloj atómico y UT está basado en un reloj astronómico.

Las fechas en Java, comienzan en el valor standar based time llamado “epoch” que hace referencia al 1 de Enero de 1970, 0 horas 0 minutos 0 segundos GMT.

La clase Date posee métodos que permiten la manipulación de fechas. La clase Date pertenece al paquete java.util del API de Java.

Método	Descripción.
Date()	Constructor que inicializa la fecha en el milisegundo más cercano a la fecha del sistema.
Date(int dia, int mes, int año)	Constructor que inicializa la fecha sumándole 1900 al año.
after(Date fecha2)	Retorna verdadero si la fecha esta después de la fecha del parámetro

before(Date fecha2)	Retorna verdadero si la fecha esta antes de la fecha del parámetro
compareTo(Date fecha)	Compara la fecha con la del parámetro. Retorna 0 si son iguales, entero negativo si el primer número es menor o entero positivo si el primer número es mayor.
equals(Object obj)	Compara el Date con el objeto del parámetro. Devuelve true si son iguales y false si no.
getDay()	Retorna el valor del día de la semana de la fecha. Ejemplo: si es lunes devuelve 0, martes 1, miércoles 2, jueves 3, viernes 4, sábado 5 y domingo 6.
getDate()	Retorna el número del día de la fecha.
getMonth()	Retorna el mes de la fecha.
getYear()	Retorna el año de la fecha.
getTime()	Retorna la fecha en milisegundos a partir del “epoch”.
setDate(int dia)	Asigna un día a la fecha.
setMonth(int mes)	Asigna un mes a la fecha.
setYear(int anio)	Asigna un año a la fecha.
setTime(long time)	Asigna la fecha en milisegundos a partir del “epoch”.
toString()	Retorna la fecha en una cadena de caracteres.

Pueden encontrar un ejemplo para descargar de Clases de Utilidad en el GIT de tu Curso.

En este proyecto hay un Main con todas las clases de utilidad previamente vistas.

EJERCICIOS DE APRENDIZAJE

Antes de comenzar con esta guía, les damos algunas recomendaciones:

Este módulo es uno de los más divertidos ya que vamos a comenzar a modelar los objetos del mundo real con el lenguaje de programación Java. Es importante tener en cuenta que entender la programación orientada a objetos lleva tiempo y sobre todo PRÁCTICA, así que, a no desesperarse, con cada ejercicio vamos a ir entendiendo un poco más cómo aplicar este paradigma.

VIDEOS: Te sugerimos ver los videos relacionados con este tema, antes de empezar los ejercicios, los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.

1. Crear una clase llamada Libro que contenga los siguientes atributos: ISBN, Título, Autor, Número de páginas, y un constructor con todos los atributos pasados por parámetro y un constructor vacío. Crear un método para cargar un libro pidiendo los datos al usuario y luego informar mediante otro método el número de ISBN, el título, el autor del libro y el numero de páginas.

2. Declarar una clase llamada Circunferencia que tenga como atributo privado el radio de tipo real. A continuación, se deben crear los siguientes métodos:

   a) Método constructor que inicialice el radio pasado como parámetro.

   b) Métodos get y set para el atributo radio de la clase Circunferencia.

   c) Método para crearCircunferencia(): que le pide el radio y lo guarda en el atributo del objeto.

   d) Método area(): para calcular el área de la circunferencia **($\mathbf{Area = \pi * radio^2}$).**

   e) Método perimetro(): para calcular el perímetro (**$\mathbf{Perimetro = 2 * \pi * radio}$).**

3. Crear una clase llamada Operacion que tenga como atributos privados numero1 y numero2. A continuación, se deben crear los siguientes métodos:

   a) Método constructor con todos los atributos pasados por parámetro.

   b) Metodo constructor sin los atributos pasados por parámetro.

   c) Métodos get y set.

   d) Método para crearOperacion(): que le pide al usuario los dos números y los guarda en los atributos del objeto.

   e) Método sumar(): calcular la suma de los números y devolver el resultado al main.

   f) Método restar(): calcular la resta de los números y devolver el resultado al main

   g) Método multiplicar(): primero valida que no se haga una multiplicación por cero, si fuera a multiplicar por cero, el método devuelve 0 y se le informa al usuario el error. Si no, se hace la multiplicación y se devuelve el resultado al main

   h) Método dividir(): primero valida que no se haga una división por cero, si fuera a pasar una división por cero, el método devuelve 0 y se le informa al usuario el error se le informa al usuario. Si no, se hace la división y se devuelve el resultado al main.

4. Crear una clase Rectángulo que modele rectángulos por medio de un atributo privado **base** y un atributo privado **altura**. La clase incluirá un método para crear el rectángulo con los datos del Rectángulo dados por el usuario. También incluirá un método para calcular la superficie del rectángulo y un método para calcular el perímetro del rectángulo. Por último, tendremos un método que dibujará el rectángulo mediante asteriscos usando la base y la altura. Se deberán además definir los métodos getters, setters y constructores correspondientes.

   Superficie = base * altura / **Perímetro** = (base + altura) * 2.

5. Realizar una clase llamada Cuenta (bancaria) que debe tener como mínimo los atributos: **numeroCuenta** (entero), el **DNI del cliente** (entero largo) y el **saldo actual** (entero). Las operaciones asociadas a dicha clase son:

   a) Constructor por defecto y constructor con DNI, saldo, número de cuenta e interés.

   b) Agregar los métodos getters y setters correspondientes

   c) Metodo para crear un objeto Cuenta, pidiéndole los datos al usuario.

   d) Método ingresar(double ingreso): el método recibe una cantidad de dinero a ingresar y se la sumara a saldo actual.

   e) Método retirar(double retiro): el método recibe una cantidad de dinero a retirar y se la restará al saldo actual. Si la cuenta no tiene la cantidad de dinero a retirar, se pondrá el saldo actual en 0.

   f) Método extraccionRapida(): le permitirá sacar solo un 20% de su saldo. Validar que el usuario no saque más del 20%.

   g) Método consultarSaldo(): permitirá consultar el saldo disponible en la cuenta.

   h) Método consultarDatos(): permitirá mostrar todos los datos de la cuenta

6. Programa Nespresso. Desarrolle una clase Cafetera con los atributos **capacidadMaxima** (la cantidad máxima de café que puede contener la cafetera) y **cantidadActual** (la cantidad actual de café que hay en la cafetera). Implemente, al menos, los siguientes métodos:

   - Constructor predeterminado o vacío
   - Constructor con la capacidad máxima y la cantidad actual
   - Métodos getters y setters.
   - Método llenarCafetera(): hace que la cantidad actual sea igual a la capacidad máxima.
   - Método servirTaza(int): se pide el tamaño de una taza vacía, el método recibe el tamaño de la taza y simula la acción de servir la taza con la capacidad indicada. Si la cantidad actual de café “no alcanza” para llenar la taza, se sirve lo que quede. El método le informará al usuario si se llenó o no la taza, y de no haberse llenado en cuanto quedó la taza.
   - Método vaciarCafetera(): pone la cantidad de café actual en cero.
   - Método agregarCafe(int): se le pide al usuario una cantidad de café, el método lo recibe y se añade a la cafetera la cantidad de café indicada.

7. Realizar una clase llamada Persona que tenga los siguientes atributos: nombre, edad, sexo ('H' hombre, 'M' mujer, 'O' otro), peso y altura. Si el alumno desea añadir algún otro atributo, puede hacerlo. Los métodos que se implementarán son:

- Un constructor por defecto.
- Un constructor con todos los atributos como parámetro.
- Métodos getters y setters de cada atributo.
- Metodo crearPersona(): el método crear persona, le pide los valores de los atributos al usuario y después se le asignan a sus respectivos atributos para llenar el objeto Persona. Además, comprueba que el sexo introducido sea correcto, es decir, H, M o O. Si no es correcto se deberá mostrar un mensaje
- Método calcularIMC(): calculara si la persona está en su peso ideal (peso en kg/(altura^2 en mt2)). Si esta fórmula da por resultado un valor menor que 20, significa que la persona está por debajo de su peso ideal y la función devuelve un -1. Si la fórmula da por resultado un número entre 20 y 25 (incluidos), significa que la persona está en su peso ideal y la función devuelve un 0. Finalmente, si el resultado de la fórmula es un valor mayor que 25 significa que la persona tiene sobrepeso, y la función devuelve un 1.
- Método esMayorDeEdad(): indica si la persona es mayor de edad. La función devuelve un booleano.

A continuación, en la clase main hacer lo siguiente:

Crear 4 objetos de tipo Persona con distintos valores, a continuación, llamaremos todos los métodos para cada objeto, deberá comprobar si la persona está en su peso ideal, tiene sobrepeso o está por debajo de su peso ideal e indicar para cada objeto si la persona es mayor de edad.

Por último, guardaremos los resultados de los métodos calcularIMC y esMayorDeEdad en distintas variables, para después en el main, calcular un porcentaje de esas 4 personas cuantas están por debajo de su peso, cuantas en su peso ideal y cuantos, por encima, y también calcularemos un porcentaje de cuantos son mayores de edad y cuantos menores.

CLASES DE UTILIDAD EN JAVA

Los métodos disponibles para las clases de utilidad Integer, Arrays y Date están en esta guía. Recordar que la clase String y Math están explicadas en la guía anterior de Intro Java.

VIDEOS: Te sugerimos ver los videos relacionados con este tema, antes de empezar los ejercicios, los podrás encontrar en tu aula virtual o en nuestro canal de YouTube.

8. Realizar una clase llamada Cadena que tenga como atributos una frase (de tipo de String) y su longitud. En el main se creará el objeto y se le pedirá al usuario que ingrese una frase que puede ser una palabra o varias palabras separadas por un espacio en blanco y a través de los métodos set, se guardará la frase y la longitud de manera automática según la longitud de la frase ingresada. Deberá además implementar los siguientes métodos:

   a) Método mostrarVocales(), deberá contabilizar la cantidad de vocales que tiene la frase ingresada.

   b) Método invertirFrase(), deberá invertir la frase ingresada y mostrarla por pantalla. Por ejemplo: Entrada: "casa blanca", Salida: "acnalb asac".

   c) Método vecesRepetido(String letra), recibirá un carácter ingresado por el usuario y contabilizar cuántas veces se repite el carácter en la frase, por ejemplo:

   d) Entrada: frase = "casa blanca". Salida: El carácter 'a' se repite 4 veces.

   e) Método compararLongitud(String frase), deberá comparar la longitud de la frase que compone la clase con otra nueva frase ingresada por el usuario.

   f) Método unirFrases(String frase), deberá unir la frase contenida en la clase Cadena con una nueva frase ingresada por el usuario y mostrar la frase resultante.

   g) Método reemplazar(String letra), deberá reemplazar todas las letras “a” que se encuentren en la frase, por algún otro carácter seleccionado por el usuario y mostrar la frase resultante.

   h) Método contiene(String letra), deberá comprobar si la frase contiene una letra que ingresa el usuario y devuelve verdadero si la contiene y falso si no.

Método Static y Clase Math

9. Realizar una clase llamada Matemática que tenga como atributos dos números reales con los cuales se realizarán diferentes operaciones matemáticas. La clase deber tener un constructor vacío, parametrizado y get y set. En el main se creará el objeto y se usará el Math.random para generar los dos números y se guardaran en el objeto con su respectivos set. Deberá además implementar los siguientes métodos:

   a) Método devolverMayor() para retornar cuál de los dos atributos tiene el mayor valor

   b) Método calcularPotencia() para calcular la potencia del mayor valor de la clase elevado al menor número. Previamente se deben redondear ambos valores.

   c) Método calculaRaiz(), para calcular la raíz cuadrada del menor de los dos valores. Antes de calcular la raíz cuadrada se debe obtener el valor absoluto del número.

Clase Arrays

10. Realizar un programa en Java donde se creen dos arreglos: el primero será un arreglo A de 50 números reales, y el segundo B, un arreglo de 20 números, también reales. El programa deberá inicializar el arreglo A con números aleatorios y mostrarlo por pantalla. Luego, el arreglo A se debe ordenar de menor a mayor y copiar los primeros 10 números ordenados al arreglo B de 20 elementos, y rellenar los 10 últimos elementos con el valor 0.5. Mostrar los dos arreglos resultantes: el ordenado de 50 elementos y el combinado de 20.

Clase Date

11. Pongamos de lado un momento el concepto de POO, ahora vamos a trabajar solo con la clase Date. En este ejercicio deberemos instanciar en el main, una fecha usando la clase Date, para esto vamos a tener que crear 3 variables, dia, mes y anio que se le pedirán al usuario para poner el constructor del objeto Date. Una vez creada la fecha de tipo Date, deberemos mostrarla y mostrar cuantos años hay entre esa fecha y la fecha actual, que se puede conseguir instanciando un objeto Date con constructor vacío.

    Ejemplo fecha: Date fecha = new Date(anio, mes, dia);

    Ejemplo fecha actual: Date fechaActual = new Date();

12. Implemente la clase Persona. Una persona tiene un nombre y una fecha de nacimiento (Tipo Date), constructor vacío, constructor parametrizado, get y set. Y los siguientes métodos:

    - Agregar un método de creación del objeto que respete la siguiente firma: crearPersona(). Este método rellena el objeto mediante un Scanner y le pregunta al usuario el nombre y la fecha de nacimiento de la persona a crear, recordemos que la fecha de nacimiento debe guardarse en un Date y los guarda en el objeto.

    - Escribir un método calcularEdad() a partir de la fecha de nacimiento ingresada. Tener en cuenta que para conocer la edad de la persona también se debe conocer la fecha actual.

    - Agregar a la clase el método menorQue(int edad) que recibe como parámetro otra edad y retorna true en caso de que el receptor tenga menor edad que la persona que se recibe como parámetro, o false en caso contrario.

    - Metodo mostrarPersona(): este método muestra la persona creada en el método anterior.

EJERCICIOS DE APRENDIZAJE EXTRA

Estos van a ser ejercicios para reforzar los conocimientos previamente vistos. Estos pueden realizarse cuando hayas terminado la guía y tengas una buena base sobre lo que venimos trabajando. Además, si ya terminaste la guía y te queda tiempo libre en las mesas, podes continuar con estos ejercicios extra, recordando siempre que no es necesario que los termines para continuar con el tema siguiente. Por último, recordá que la prioridad es ayudar a los compañeros de la mesa y que cuando tengas que ayudar, lo más valioso es que puedas explicar el ejercicio con la intención de que tu compañero lo comprenda, y no sólo mostrarlo. ¡Muchas gracias!

1. Desarrollar una clase Cancion con los siguientes atributos: titulo y autor. Se deberá definir además dos constructores: uno vacío que inicializa el titulo y el autor a cadenas vacías y otro que reciba como parámetros el titulo y el autor de la canción. Se deberán además definir los métodos getters y setters correspondientes.

2. Definir una clase llamada Puntos que contendrá las coordenadas de dos puntos, sus atributos serán, x1, y1, x2, y2, siendo cada x e y un punto. Generar un objeto puntos usando un método crearPuntos() que le pide al usuario los dos números y los ingresa en los atributos del objeto. Después, a través de otro método calcular y devolver la distancia que existe entre los dos puntos que existen en la clase Puntos. Para conocer como calcular la distancia entre dos puntos consulte el siguiente link:

   http://www.matematicatuya.com/GRAFICAecuaciones/S1a.html

3. Vamos a realizar una clase llamada Raices, donde representaremos los valores de una ecuación de 2o grado. Tendremos los 3 coeficientes como atributos, llamémosles a, b y c. Hay que insertar estos 3 valores para construir el objeto a través de un método constructor. Luego, las operaciones que se podrán realizar son las siguientes:

   - Método getDiscriminante(): devuelve el valor del discriminante (double). El discriminante tiene la siguiente formula: (b^2)-4*a*c
   - Método tieneRaices(): devuelve un booleano indicando si tiene dos soluciones, para que esto ocurra, el discriminante debe ser mayor o igual que 0.
   - Método tieneRaiz(): devuelve un booleano indicando si tiene una única solución, para que esto ocurra, el discriminante debe ser igual que 0.
   - Método obtenerRaices(): llama a tieneRaíces() y si devolvió true, imprime las 2 posibles soluciones.
   - Método obtenerRaiz(): llama a tieneRaiz() y si devolvió true imprime una única raíz. Es en el caso en que se tenga una única solución posible.
   - Método calcular(): esté método llamará tieneRaices() y a tieneRaíz(), y mostrará por pantalla las posibles soluciones que tiene nuestra ecuación con los métodos obtenerRaices() o obtenerRaiz(), según lo que devuelvan nuestros métodos y en caso de no existir solución, se mostrará un mensaje.

Nota: Formula ecuación 2o grado: (-b±√((b^2)-(4*a*c)))/(2*a) Solo varia el signo delante de -b

4. Dígito Verificador. Crear una clase NIF que se usará para mantener DNIs con su correspondiente letra (NIF). Los atributos serán el número de DNI (entero largo) y la letra (String o char) que le corresponde. Dispondrá de los siguientes métodos:

- Métodos getters y setters para el número de DNI y la letra.
- Método crearNif(): le pide al usuario el DNI y con ese DNI calcula la letra que le corresponderá. Una vez calculado, le asigna la letra que le corresponde según el resultado del calculo.
- Método mostrar(): que nos permita mostrar el NIF (ocho dígitos, un guion y la letra en mayúscula; por ejemplo: 00395469-F).

La letra correspondiente al dígito verificador se calculará a traves de un método que funciona de la siguiente manera: Para calcular la letra se toma el resto de dividir el número de DNI por 23 (el resultado debe ser un número entre 0 y 22). El método debe buscar en un array (vector) de caracteres la posición que corresponda al resto de la división para obtener la letra correspondiente. La tabla de caracteres es la siguiente:

POSICIÓN	LETRA
0	T
1	R
2	W
3	A
4	G
5	M
6	Y
7	F
8	P
9	D
10	X
11	B
12	N
13	J
14	Z
15	S
16	Q
17	V
18	H
19	L
20	C
21	K
22	E

5. Crea una clase en Java donde declares una variable de tipo array de Strings que contenga los doce meses del año, en minúsculas. A continuación, declara una variable mesSecreto de tipo String, y hazla igual a un elemento del array (por ejemplo, mesSecreto = mes[9]. El programa debe pedir al usuario que adivine el mes secreto. Si el usuario acierta mostrar un mensaje, y si no lo hace, pedir que vuelva a intentar adivinar el mes secreto. Un ejemplo de ejecución del programa podría ser este:

Adivine el mes secreto. Introduzca el nombre del mes en minúsculas: febrero

No ha acertado. Intente adivinarlo introduciendo otro mes: agosto

¡Ha acertado!

6. Juego Ahorcado: Crear una clase Ahorcado (como el juego), la cual deberá contener como atributos, un vector con la palabra a buscar, la cantidad de letras encontradas y la cantidad jugadas máximas que puede realizar el usuario. Definir los siguientes métodos con los parámetros que sean necesarios:

- Constructores, tanto el vacío como el parametrizado.
- Metodo crearJuego(): le pide la palabra al usuario y cantidad de jugadas máxima. Con la palabra del usuario, pone la longitud de la palabra, como la longitud del vector. Después ingresa la palabra en el vector, letra por letra, quedando cada letra de la palabra en un índice del vector. Y también, guarda en cantidad de jugadas máximas, el valor que ingresó el usuario y encontradas en 0.
- Método longitud(): muestra la longitud de la palabra que se debe encontrar. Nota: buscar como se usa el vector.length.
- Método buscar(letra): este método recibe una letra dada por el usuario y busca sila letra ingresada es parte de la palabra o no. También informará si la letra estaba o no.
- Método encontradas(letra): que reciba una letra ingresada por el usuario y muestre cuantas letras han sido encontradas y cuantas le faltan. Este método además deberá devolver true si la letra estaba y false si la letra no estaba, ya que, cada vez que se busque una letra que no esté, se le restará uno a sus oportunidades.
- Método intentos(): para mostrar cuantas oportunidades le queda al jugador.
- Método juego(): el método juego se encargará de llamar todos los métodos previamente mencionados e informará cuando el usuario descubra toda la palabra o se quede sin intentos. Este método se llamará en el main.

Un ejemplo de salida puede ser así:

Ingrese una letra:

a

Longitud de la palabra: 6

Mensaje: La letra pertenece a la palabra

Número de letras (encontradas, faltantes): (3,4)

Número de oportunidades restantes: 4

Ingrese una letra:

z

Longitud de la palabra: 6

Mensaje: La letra no pertenece a la palabra

Número de letras (encontradas, faltantes): (3,4)

Número de oportunidades restantes: 3

Ingrese una letra:

b

Longitud de la palabra: 6

Mensaje: La letra no pertenece a la palabra

Número de letras (encontradas, faltantes): (4,3)

Número de oportunidades restantes: 2

Ingrese una letra:

u

Longitud de la palabra: 6

Mensaje: La letra no pertenece a la palabra

Número de letras (encontradas, faltantes): (4,3)

Número de oportunidades restantes: 1

Ingrese una letra:

q

Longitud de la palabra: 6

Mensaje: La letra no pertenece a la palabra

Mensaje: Lo sentimos, no hay más oportunidades

Pueden encontrar un ejemplo para descargar de vector en el GIT de tu Curso.

BIBLIOGRAFÍA

Información sacada de libros:

- Fundamentos de Programación de Luis Joyanes Aguilar

- Fundamentos de Programación Java de Jorge Martínez Ladrón de Guevara en conjunto con la Facultad de Informática (Universidad Complutense de Madrid).

- Introducción a la programación Java de John S. Dean y Raymond H. Dean

Información sacada de las páginas:

- https://profile.es/blog/que-son-los-paradigmas-de-programacion/

- https://profile.es/blog/que-es-la-programacion-orientada-a-objetos/