



Ministerio de Producción  
Presidencia de la Nación

Ministerio de Educación y Deportes

Subsecretaría de Servicios Tecnológicos y Productivos



Programa  
**111**  
**mil**  
VOS PODÉS  
SER UNO.

**Clase e instancia**

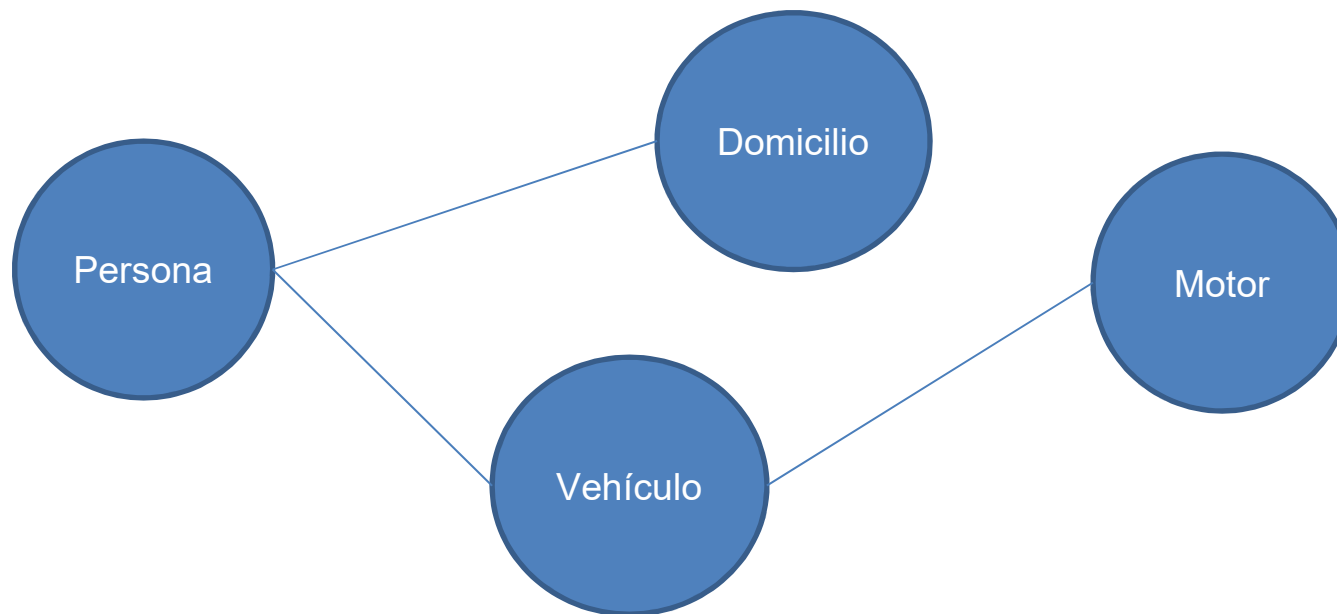
# Programación

- Clase vs. Instancia
- Tipos primitivos
- Atributos
- Métodos
- Referencia
- Ejecución



# Clase vs. Instancia

Java es un lenguaje de Programación Orientado a **Objetos** (POO). Esto significa que la organización de los programas Java está centrada en el concepto de **Objeto**.



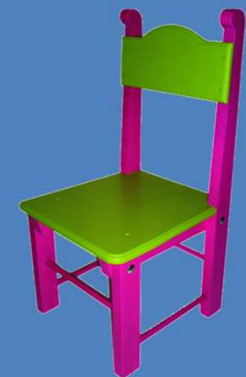
# Clase vs. Instancia

¿Qué es un **Objeto**?

- Un **Objeto** es una entidad de software que tiene asociado un conjunto de atributos y comportamiento.
- Los **Objetos** son creados a partir de moldes llamados **Clases**.
- Una **Clase** que especifican cuales atributos y que comportamiento está asociado a cada **Objeto** o **Instancia** de esa clase.

Clase: Silla

Instancias:



# Clase vs. Instancia

- **Clase:** las clases se definen mediante código fuente, el cual debe ser escrito antes de ejecutar el programa.
- **Instancia:** las instancias se crean cuando el programa está ejecutando (runtime) utilizando como molde las **Clases** predefinidas

Persona
- dni: int - nombre: String - apellido: String - sueldo: int
+ Persona(int, String, String, int) + getSuelo(): int + puedePrestamo(int, int): boolean + setSuelo(int): void

personal : Persona

private int dni	29023023	Inspect
private String nombre	"Alicia"	Get
private String apellido	"Harris"	
private int sueldo	20000	

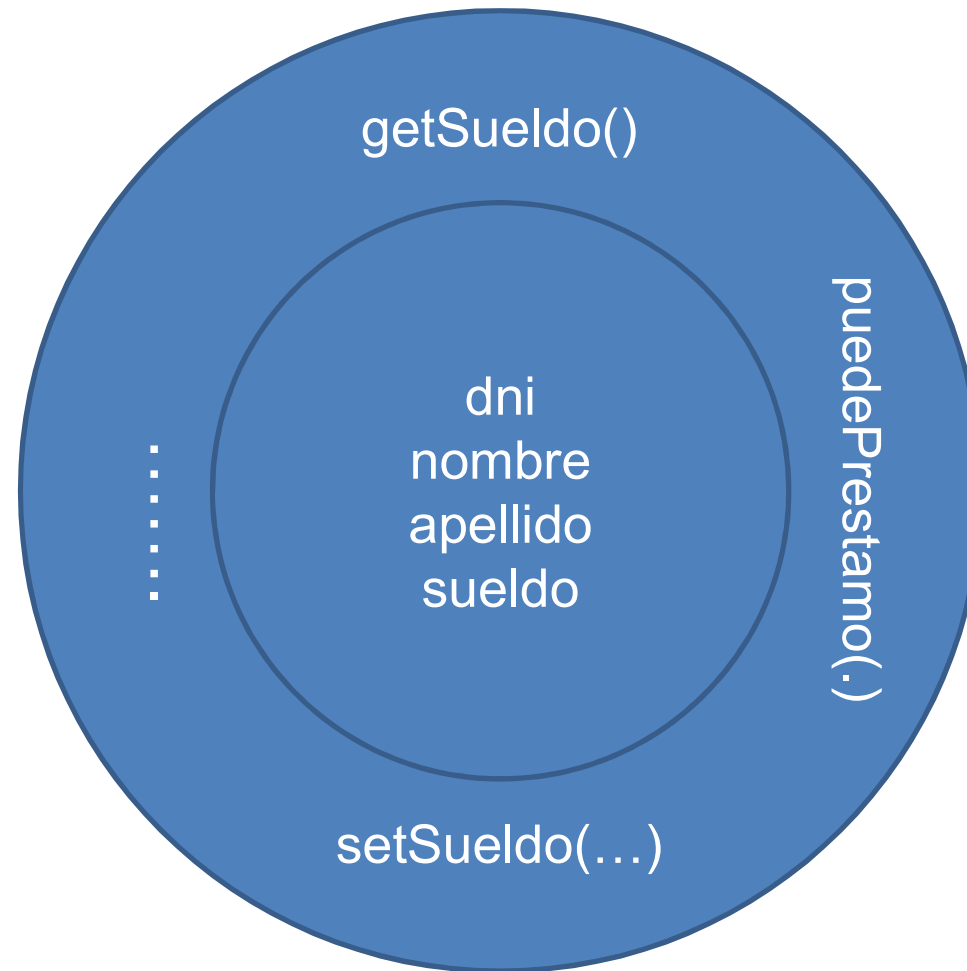
Show static fields Close

persona2 : Persona

private int dni	34856521	Inspect
private String nombre	"Cristian"	Get
private String apellido	"Romero"	
private int sueldo	15000	

Show static fields Close

# Clase Persona



# Clase Persona

Definición de paquete. Los paquetes son la forma de organizar las clases, agrupándolas por afinidad

```
package edu.unicen.poo111mil;
```

Definición de la clase:

- public: puede ser vista por cualquier clase del programa
- class: estamos definiendo una clase.
- Persona: el nombre de nuestra clase.

```
public class Persona {  
    private int dni;  
    private String nombre;  
    private String apellido;  
    private int sueldo;
```

```
    public Persona(int dni, String nombre,  
                   String apellido, int sueldo) {  
        this.dni = dni;  
        this.nombre = nombre;  
        this.apellido = apellido;  
        this.sueldo = sueldo;  
    }
```

...

# Clase Persona

Definición de los atributos:

- private: significa que solo pueden ser vistos desde la clase y sus instancias.
- int/String: define el tipo de dato que contiene el atributo. String es una cadena de texto, int son números enteros\*.

```
package edu.unicen.poo111mil;
```

```
public class Persona {  
    private int dni;  
    private String nombre;  
    private String apellido;  
    private int sueldo;
```

Definición de un constructor:

- Sirve para crear instancia de nuestra clase.
- Recibe como parámetro: dni, nombre, apellido y sueldo.

```
    public Persona(int dni, String nombre,  
                  String apellido, int sueldo) {  
        this.dni = dni;  
        this.nombre = nombre;  
        this.apellido = apellido;  
        this.sueldo = sueldo;
```

this.dni vs dni:

- Como el constructor recibe un parámetro que se llama igual que un atributo hay que desambiguar.
- this: significa esta instancia. Es decir, la instancia que se está creando.

```
    }
```

```
    ...
```



# Clase Persona

Métodos: definen el comportamiento de las instancias de la clase. Los primeros dos métodos son conocidos como getters/setters. Se utilizan para que otras clases puedan acceder a los atributos de instancias de esta clase.

El this. no es necesario en ninguno de estos métodos porque no hay ambigüedad, sin embargo el código es más claro si está presente.

```
public int getSueldo(){  
    return this.sueldo;  
}
```

Instrucción return: termina la ejecución del método y retorna lo que se encuentra a su derecha.

```
public void setSueldo(int nuevoSueldo){  
    this.sueldo = nuevoSueldo;  
}
```

```
public boolean puedePrestamo(int cant, int meses){  
    return (cant/meses) < this.sueldo;  
}
```

Métodos:

- Modificador de acceso (public): tipo de acceso.
- Tipo de retorno: define qué devuelve el método. Un caso especial es void, significa que el método no retorna ningún dato.
- Nombre del método.
- Parámetros: define qué datos recibe el método y como se llaman.

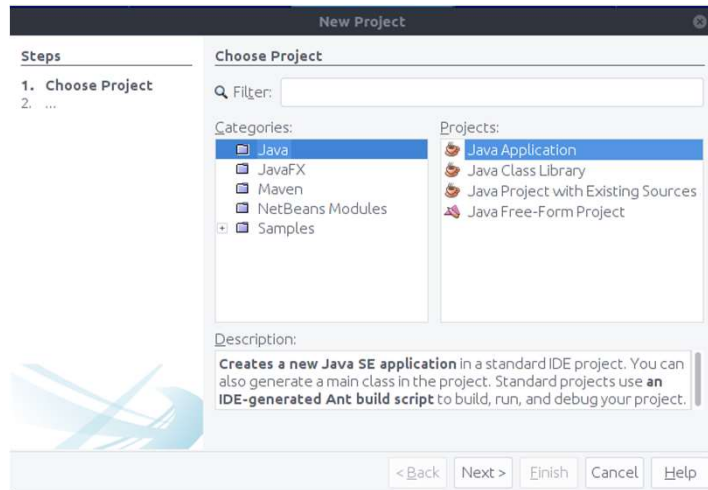
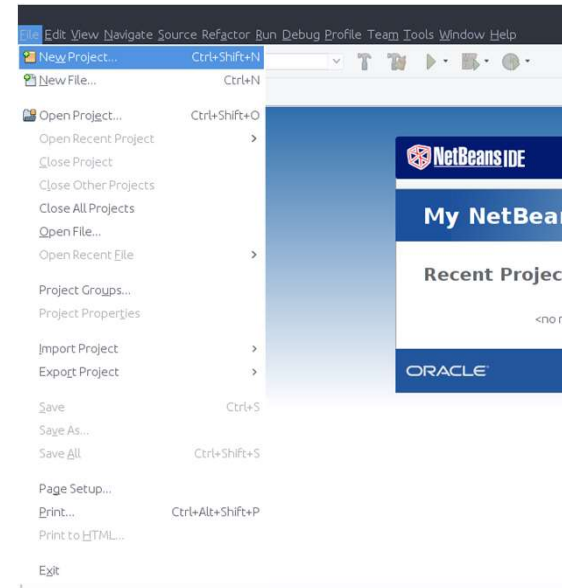


# Clase Persona: ejercicio práctico

- Crear un nuevo proyecto en NetBeans.
- Crear la clase Persona en su correspondiente paquete.
- Crear una clase con un punto de entrada Java. Es decir un método con la signatura `public static void main(String[] args)`
- Crear 2 instancias de la clase persona. 34.233.434, Pedro Fernández, sueldo: 23000 y 32.242.121, Clara García, sueldo: 25000.
- Imprimir sus atributos por pantalla.
- Verificar si pueden pedir un préstamo de 125.000 a 5 meses. Imprimir el resultado por pantalla.
- Ejecutar el programa.

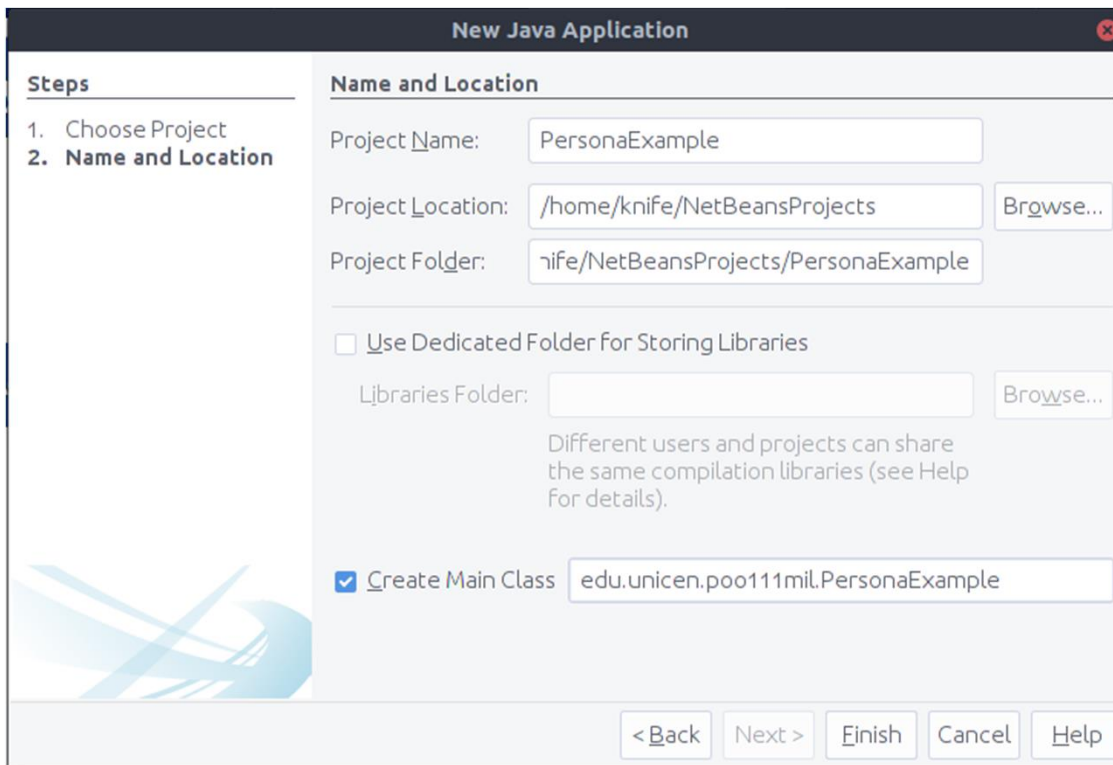
# Clase Persona: creando el proyecto

- Abrir NetBeans.
- Ir al menú “File” (Archivo).
- Seleccionar la opción “New Project” (Nuevo Proyecto)



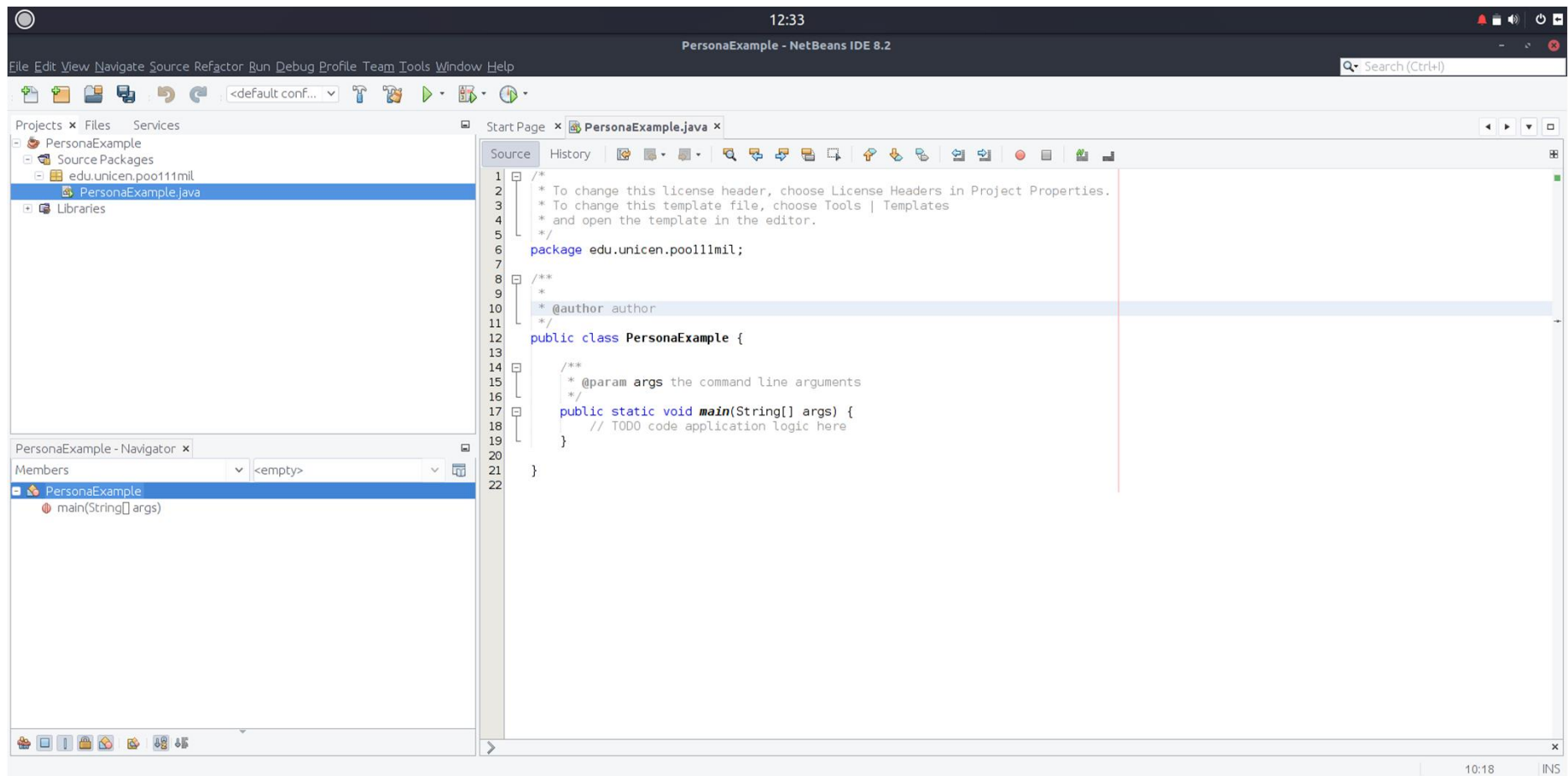
- En “Categories”, seleccionar “Java”
- En “Project”, seleccionar “Java Application”
- Hacer clic en “Next >”.

# Clase Persona: creando el proyecto



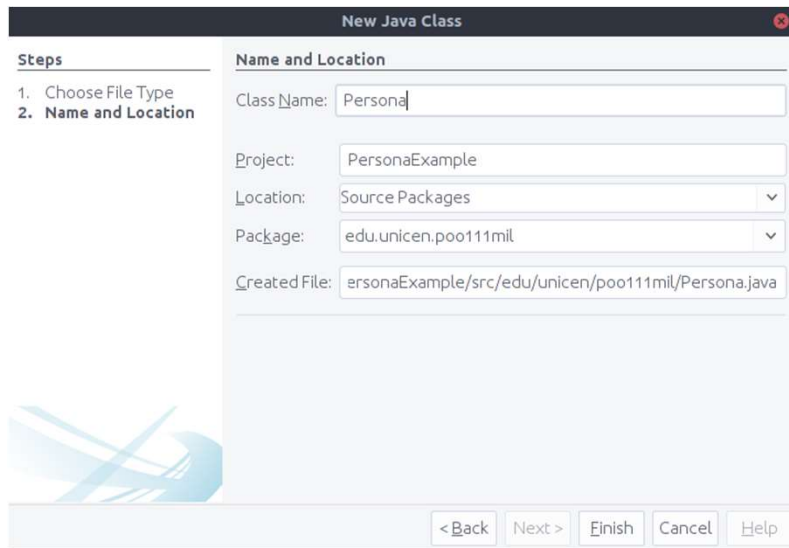
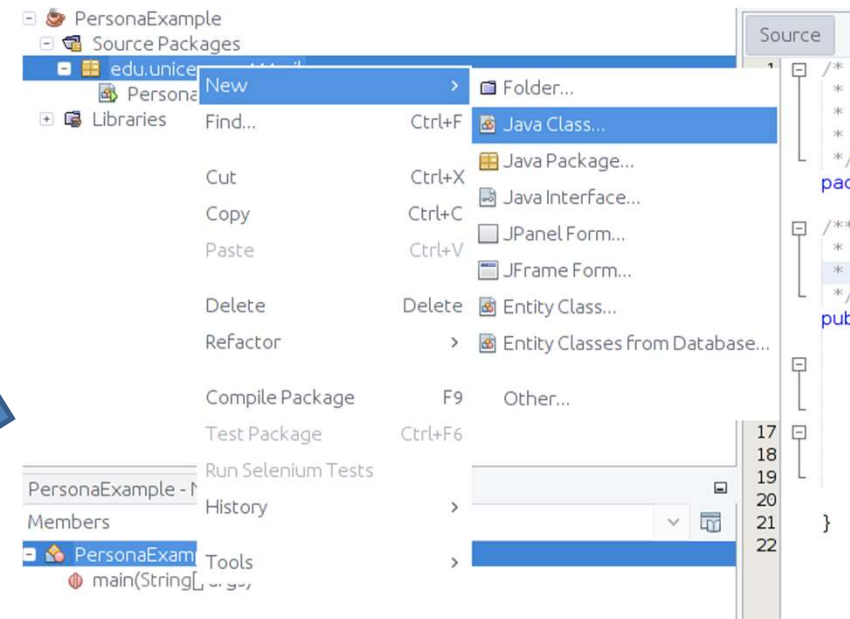
- Project Name: Nombre de nuestro proyecto. En este caso “PersonaExample”
- Project Location: En que lugar se guardan nuestros proyectos.
- Project Folder: En que lugar se guardará este proyecto.
- Create Main Class: Si deseamos crear la clase main. Completar con nombre del paquete y de la clase.

# Clase Persona: creando el proyecto



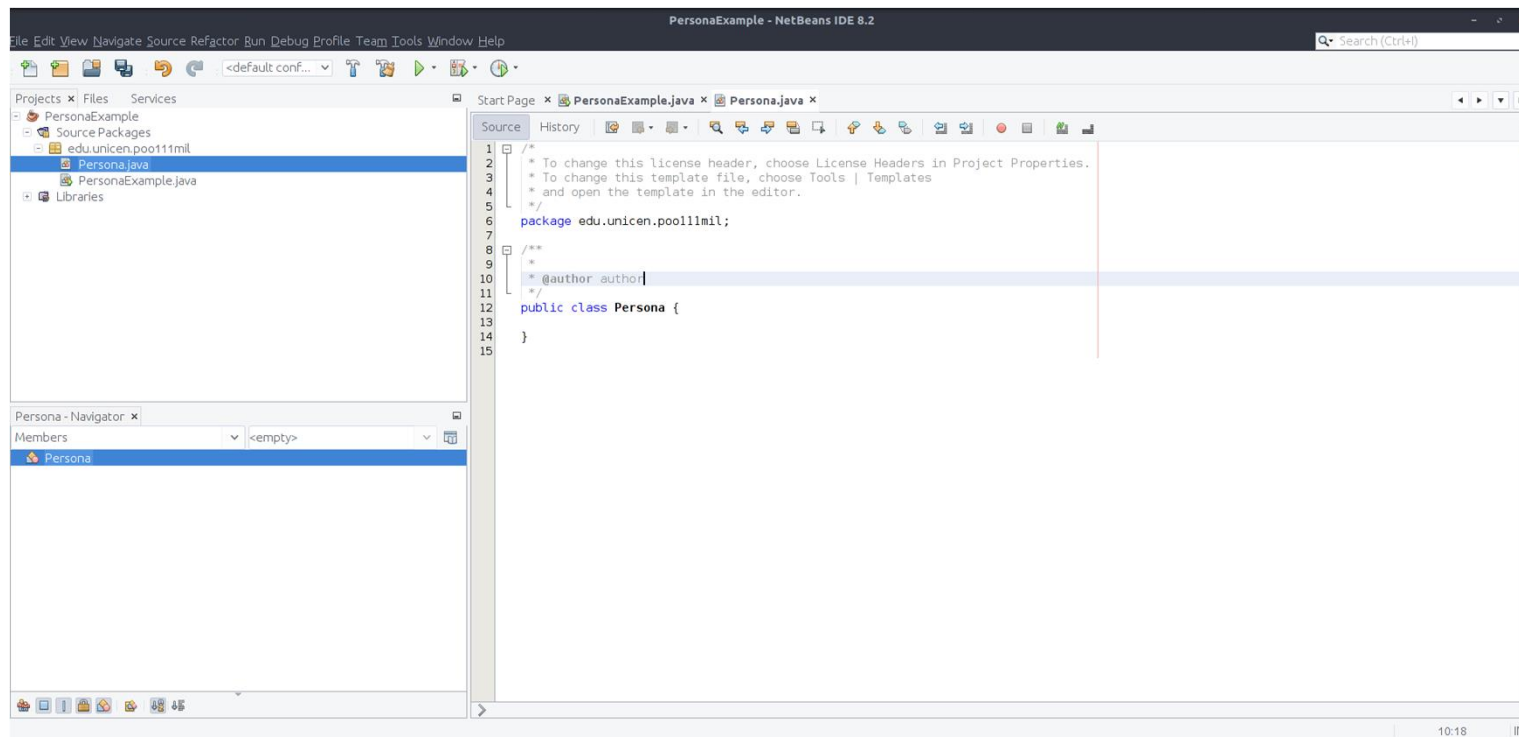
# Clase Persona: creando la clase

- En la pestaña de la izquierda, hacer clic secundario sobre el nombre del paquete.
- Seleccionar “New” > “Java Class...”



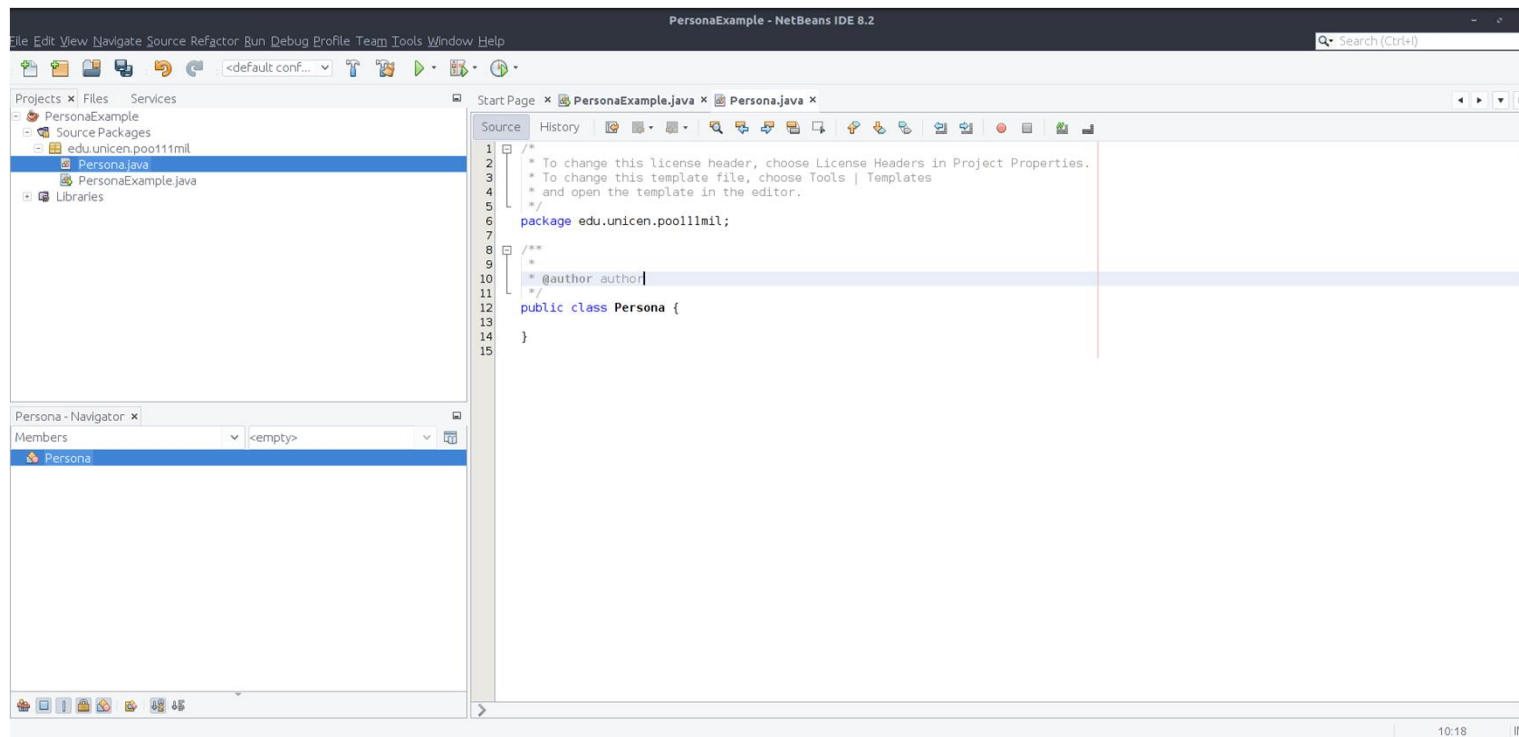
- Completar el Nombre de la clase.
- El paquete y las otras propiedades vienen por defecto.
- Hacer clic en “Finish”.

# Clase Persona: creando la clase





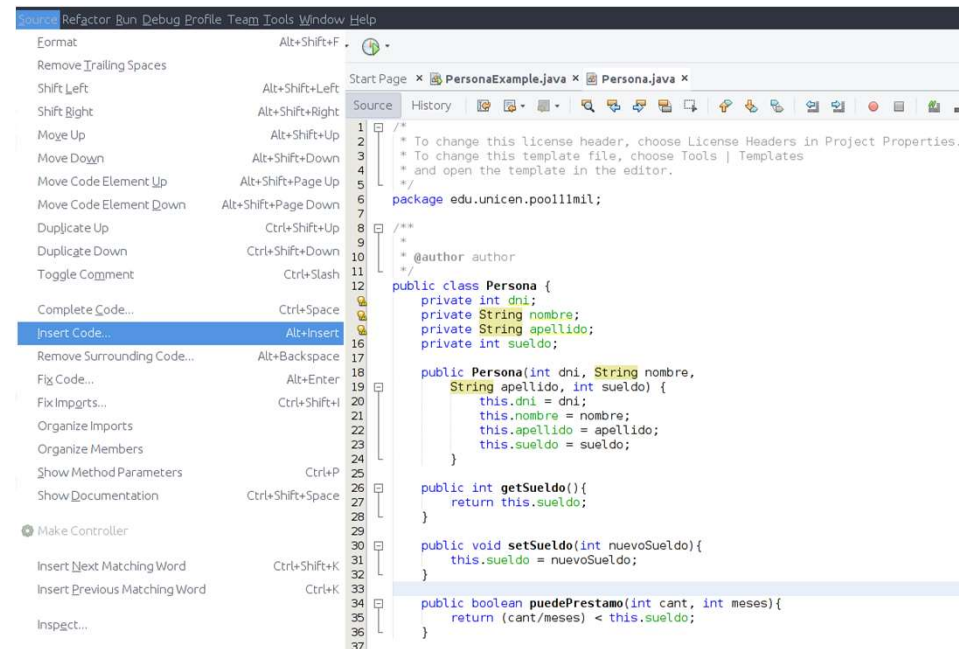
# Clase Persona: creando la clase





# Clase Persona: creando la clase

- Empezamos a codificar...
- Pero, los getters y setters son algo muy común de hacer y NetBeans facilita esta tarea.
- Seleccionar en el menú “Source” (Código fuente) > “Insert code” (Insertar código) u oprimir las teclas “Alt+Insert” (Alt+Insertar).



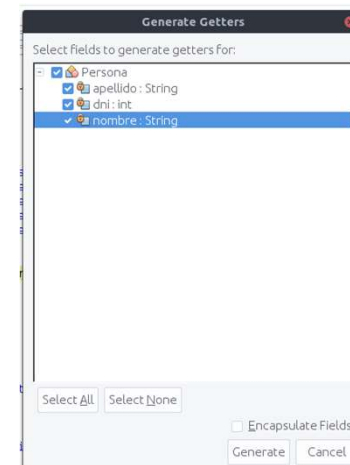
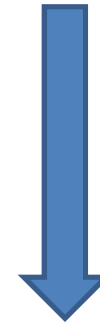
```
Source Refactor Run Debug Profile Team Tools Window Help
Format Alt+Shift+F
Remove Trailing Spaces
Shift Left Alt+Shift+Left
Shift Right Alt+Shift+Right
Move Up Alt+Shift+Up
Move Down Alt+Shift+Down
Move Code Element Up Alt+Shift+Page Up
Move Code Element Down Alt+Shift+Page Down
Duplicate Up Ctrl+Shift+Up
Duplicate Down Ctrl+Shift+Down
Toggle Comment Ctrl+/
Complete Code... Ctrl+Space
Insert Code... Alt+Insert
Remove Surrounding Code... Alt+Backspace
Fix Code... Alt+Enter
Organize Imports Ctrl+Shift+I
Organize Members
Show Method Parameters Ctrl+P
Show Documentation Ctrl+Shift+Space
Make Controller
Insert Next Matching Word Ctrl+Shift+K
Insert Previous Matching Word Ctrl+K
Inspect...

1  /*
2  * To change this license header, choose License Headers in Project Properties.
3  * To change this template file, choose Tools | Templates
4  * and open the template in the editor.
5  */
6  package edu.unicen.poo11mil;
7
8  /**
9   *
10   * @author author
11   */
12  public class Persona {
13      private int dni;
14      private String nombre;
15      private String apellido;
16      private int sueldo;
17
18      public Persona(int dni, String nombre,
19                    String apellido, int sueldo) {
20          this.dni = dni;
21          this.nombre = nombre;
22          this.apellido = apellido;
23          this.sueldo = sueldo;
24      }
25
26      public int getSueldo(){
27          return this.sueldo;
28      }
29
30      public void setSueldo(int nuevoSueldo){
31          this.sueldo = nuevoSueldo;
32      }
33
34      public boolean puedePrestamo(int cant, int meses){
35          return (cant/meses) < this.sueldo;
36      }
37  }
```

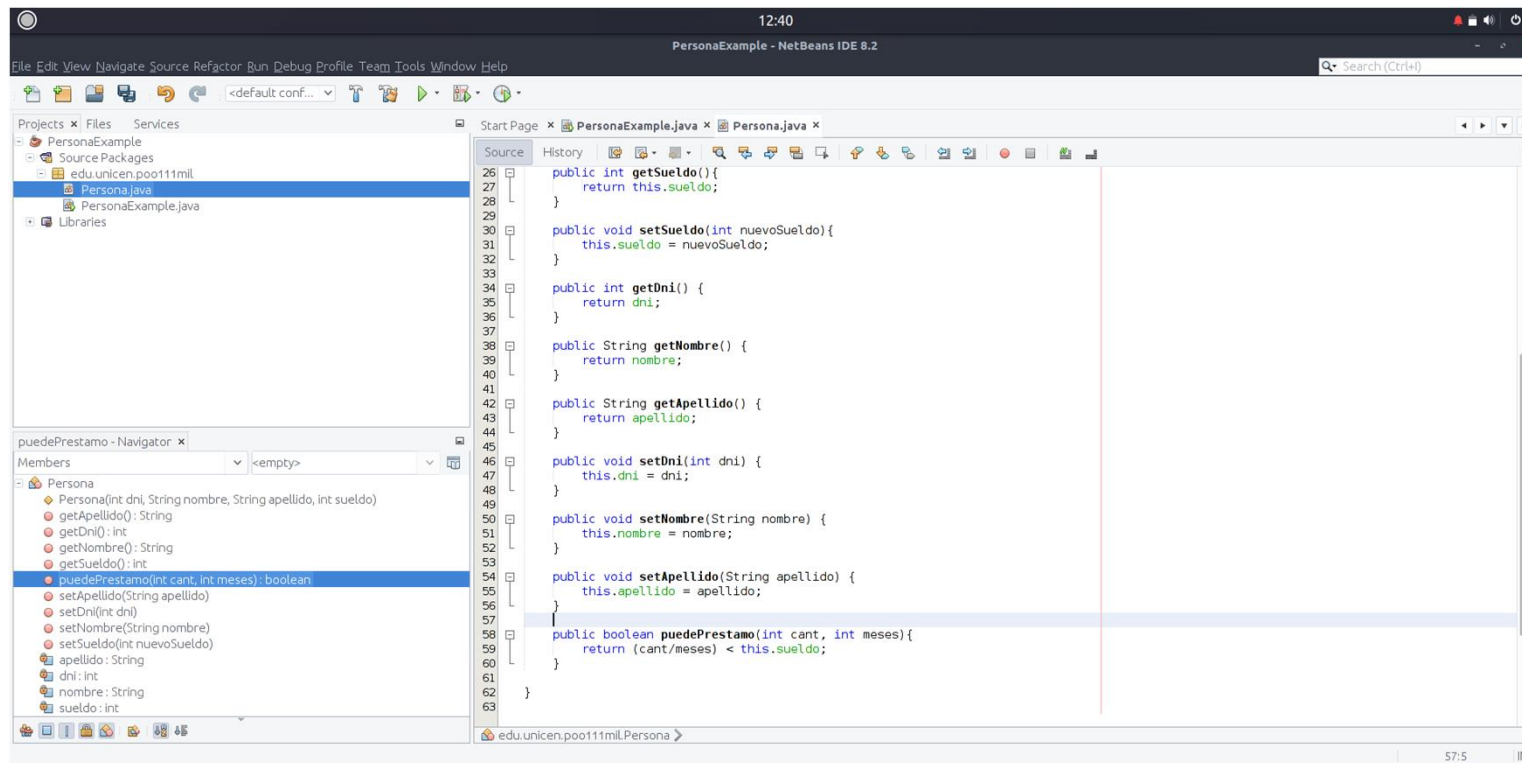
# Clase Persona: creando la clase

- Aparece un menú contextual donde está el cursor.
- Tiene las opciones más comunes como: Constructor, Getter, Setter, Getter and Setter, etc.
- Para el ejemplo, seleccionamos Getter. (Se pueden hacer getters y setter todo junto, pero lo hacemos de a uno para practicar)

```
public Persona(int dni, String nombre,  
String apellido, int sueldo) {  
    ;  
    nombre;  
    = apellido;  
    sueldo;  
}  
  
{  
    equals() and hashCode()...  
};  
  
int nuevoSueldo){  
    yoSueldo;  
}
```



# Clase Persona: creando la clase



Se generaron los getters. Ahora repita el procedimiento seleccionando setters para generar los setters.



# Clase Persona: ejercicio práctico

- ✓ Crear un nuevo proyecto en NetBeans.
- ✓ Crear la clase Persona en su correspondiente paquete.
- ✓ Crear una clase con un punto de entrada Java. Es decir un método con la signatura "public static void main(String[] args)."
- Crear 2 instancias de la clase persona. 34.233.434, Pedro Fernández, sueldo: 23000 y 32.242.121, Clara García, sueldo: 25000.
- Imprimir sus atributos por pantalla.
- Verificar si pueden pedir un préstamo de 125.000 a 5 meses. Imprimir el resultado por pantalla.
- Ejecutar el programa.

# Clase Persona: definiendo el método main

```
public static void main(String[] args) {  
    Persona pedro = new Persona(34_233_434, "Pedro", "Fernández", 23000);  
    Persona alicia = new Persona(32_242_121, "Clara", "García", 25000);  
    ...  
}
```

- Definimos dos variables, pedro y alicia de tipo persona. Las variables nos sirven para tener referencia a nuestras instancias, pero de por sí no son las instancias.
- Creamos las instancias llamando al constructor. La palabra clave “new” indica que se debe crear una nueva instancia de algo, mientras que el nombre del constructor y los parámetros indican cómo crear esa nueva instancia.

# Clase Persona: definiendo el método main

```
public static void main(String[] args) {  
    Persona pedro = new Persona(34_233_434, "Pedro", "Fernández", 23000);  
    Persona alicia = new Persona(32_242_121, "Clara", "García", 25000);  
  
    System.out.println("Nombre " + pedro.getApellido());  
    System.out.println("Apellido " + pedro.getDni());  
    System.out.println("DNI: " + pedro.getDni());  
    System.out.println("Sueldo " + pedro.getSueldo());  
    System.out.println("Puede pedir préstamo " +  
        pedro.puedePrestamo(120000, 5));  
}
```

- System.out.println(...) sirve para imprimir por pantalla.
- Cualquier cosa entre “comillas” es un String, la operación suma (+) significa concatenar.
- Después de eso, se invoca a diferentes métodos sobre el objeto referenciado por la variable llamada pedro. Se utiliza la notación variable.método(...) para indicar sobre qué objeto y qué método se está invocando.
- Invocar un método sobre una instancia también es conocido como pasaje de mensajes. Por ejemplo, pedro.getDni() es pasarle el mensaje getDni() a pedro y pedro nos retorna un mensaje con su DNI.
- ¿Qué pasa si en los System.out.println(...) cambio a pedro por alicia?

# Clase Persona: definiendo el método main

```
public static void main(String[] args) {  
    Persona pedro = new Persona(34_233_434, "Pedro", "Fernández", 23000);  
    Persona alicia = new Persona(32_242_121, "Clara", "García", 25000);  
  
    System.out.println("Nombre " + pedro.getApellido());  
    System.out.println("Apellido " + pedro.getDni());  
    System.out.println("DNI: " + pedro.getDni());  
    System.out.println("Sueldo " + pedro.getSueldo());  
    System.out.println("Puede pedir préstamo " +  
        pedro.puedePrestamo(120000, 5));  
  
    System.out.println("Nombre " + alicia.getApellido());  
    System.out.println("Apellido " + alicia.getDni());  
    System.out.println("DNI: " + alicia.getDni());  
    System.out.println("Sueldo " + alicia.getSueldo());  
    System.out.println("Puede pedir préstamo " +  
        alicia.puedePrestamo(120000, 5));  
}
```

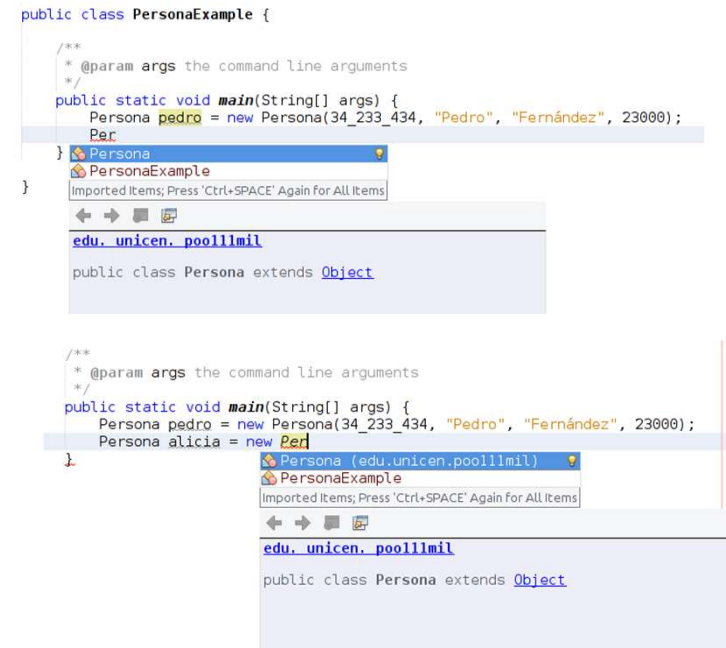
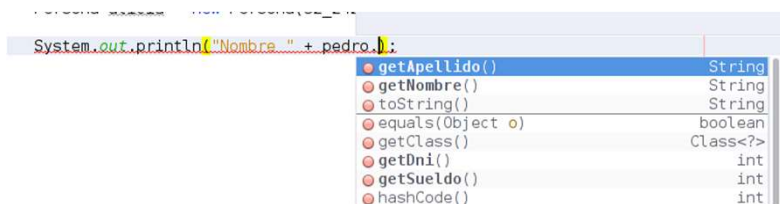
Así debe quedar nuestro programa terminado.



# Clase Persona: main en Netbeans

Hay tareas repetitivas en la programación y NetBeans nos ayuda.

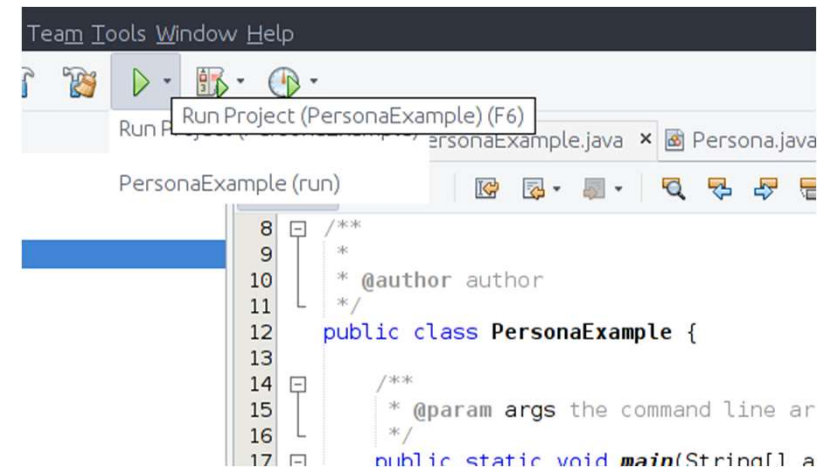
- A medida que escribimos código podemos presionar las teclas “Ctrl+espacio” y aparecen menús contextuales sugiriendo como se podría completar ese código.
- Además existen atajos, como escribir “sout” y presionar “Ctrl+espacio” para generar automáticamente el código “System.out.println()”



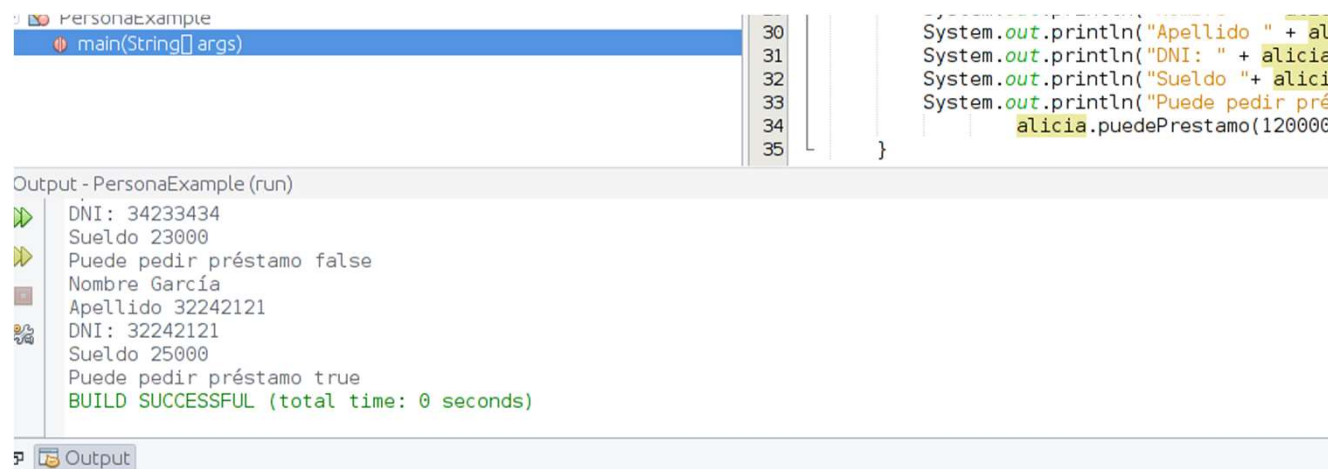


# Clase Persona: definiendo el método main

- Hacer clic en el botón de play. En caso de haber más de un proyecto, se puede seleccionar cual ejecutar haciendo clic en la flecha pequeña que apunta hacia abajo al lado del botón play.
- En la parte inferior de NetBeans aparece una “consola” donde se imprime la salida del programa.



```
8  /**
9  *
10 * @author author
11 */
12 public class PersonaExample {
13
14     /**
15     * @param args the command line ar
16     */
17     public static void main(String[] a
```



```
PersonaExample
main(String[] args)

30
31
32
33
34
35

System.out.println("Apellido " + al
System.out.println("DNI: " + alicia
System.out.println("Sueldo " + alici
System.out.println("Puede pedir pré
    alicia.puedePrestamo(120000
}

Output - PersonaExample (run)
DNI: 34233434
Sueldo 23000
Puede pedir préstamo false
Nombre García
Apellido 32242121
DNI: 32242121
Sueldo 25000
Puede pedir préstamo true
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Tipos primitivos vs. Clases

- En Java existen los tipos primitivos de datos y las clases.
- Las clases, como se vio hasta el momento, integran atributos y comportamiento. Un ejemplo de clase, es la clase Persona. String también es una clase, aunque esta clase, como muchas otras, es provista por Java. La manera de trabajar con las clases es a través de pasaje de mensajes, es decir, invocando sus métodos.
- Los tipos primitivos de datos solo contienen información. Ejemplos de tipos primitivos son boolean e int. Sobre los tipos primitivos solo se puede aplicar operadores como suma (+), resta (-), multiplicación (\*) y división (/).
- No todas las operaciones aceptan todos los operadores, por ejemplo el operador suma no puede ser aplicado sobre boolean, mientras que el operador “y lógico” (&&) no tiene sentido sobre int.
- Excepcionalmente, String soporta el operador concatenación (+), sin embargo el compilador traduce esto a objetos y pasaje de mensajes. Ninguna otra clase soporta operadores. Finalmente, el operador == está permitido para todos los tipos primitivos y clases.

# Tipos primitivos

Tipo	Tamaño	Defecto	Rango	Wrapper
byte	8 bits	0	-128, 127	Byte
short	16 bits	0	-32.768, 32.767	Short
int	32 bits	0	$-2^{32}, 2^{32}-1$	Integer
long	64 bits	0	$-2^{64}, 2^{64}-1$	Long
float	32 bits	0.0f	32-bit IEEE 754	Float
double	64 bits	0.0d	64-bit IEEE 754	Double
boolean	no-definido	false	true/false	Boolean
char	16 bits	'\u0000'	'\u0000', '\uffff'	Character



# Operadores unarios

- ++ y --: incrementa/decrementa en uno la variable.
  - var++: lee el valor de la variable y luego incrementa en uno la variable;
  - ++var: incrementa en uno la variable y luego lee el valor;
- + y -: indican si el valor es positivo o negativo.
  - +1 o +var: equivale al valor.
  - -1 o -var: equivale al valor complementario en la suma.
- ~: operación not bitwise. (está por completitud, pero queda fuera del curso)
- !: operación not para valores boolean.

# Operadores binarios

- $*$ ,  $/$  y  $\%$ : multiplicación, división y resto.
  - $2 * 3$ : resulta en 6;
  - $5 / 3$ : resulta en 1;
  - $5 \% 3$ : resulta en 2;
- $+$  y  $-$ : suma y resta.
  - $2 + 3$ : resulta en 5;
  - $2 - 3$ : resulta en -1;

Los siguientes operadores retornan valores de tipo boolean.

- $<$ : menor.
  - $1 < 2$ : retorna true.
  - $2 < 1$ : retorna false.
- $>$ : mayor.
- $<=$ : menor o igual.
- $>=$ : mayor o igual.
- `instanceof`: es instancia de.
  - `var instanceof [Class|Interface]`: retorna true si la clase de var es, hereda o implementa el segundo operando;
  - `null instanceof X`: retorna false.
- $\&\&$ : operador lógico “y”. Retorna true solo si ambos operandos son true.
- $\|\|$ : operador lógico “o”. Retorna true si al menos uno de los operandos es true.



# Operadores de igualdad

Retornan valores de tipo boolean.

- `==`: retorna true si los operandos son iguales. En el caso de objetos, este operador compara que sean la referencia al mismo objeto.
  - `10 == 10`: retorna true.
  - `a == a`: retorna true.
  - `null == null`: retorna true.
  - `Double.NaN == Double.NaN`: retorna false.
  - `(new Integer(10)) == (new Integer(10))`: retorna false.
- `!=`: retorna false si los operandos son iguales.

# Operadores de Asignación

- =: asigna el resultado de la expresión de la derecha a la variable a su izquierda.

```
Persona pedro = new Persona(34_233_434, "Pedro", "Fernández", 23000);  
int a = 10; //a vale 10
```

- +=: suma el resultado de la expresión de la derecha a la variable a su izquierda y se lo asigna.

```
a += 10 //a vale 20
```

- =: resta el resultado de la expresión de la derecha a la variable a su izquierda y se lo asigna.

- \*= /= %= &=:e n general aplican al operación a la izquierda del = (var op (exp)) y asigna el valor a la variable.

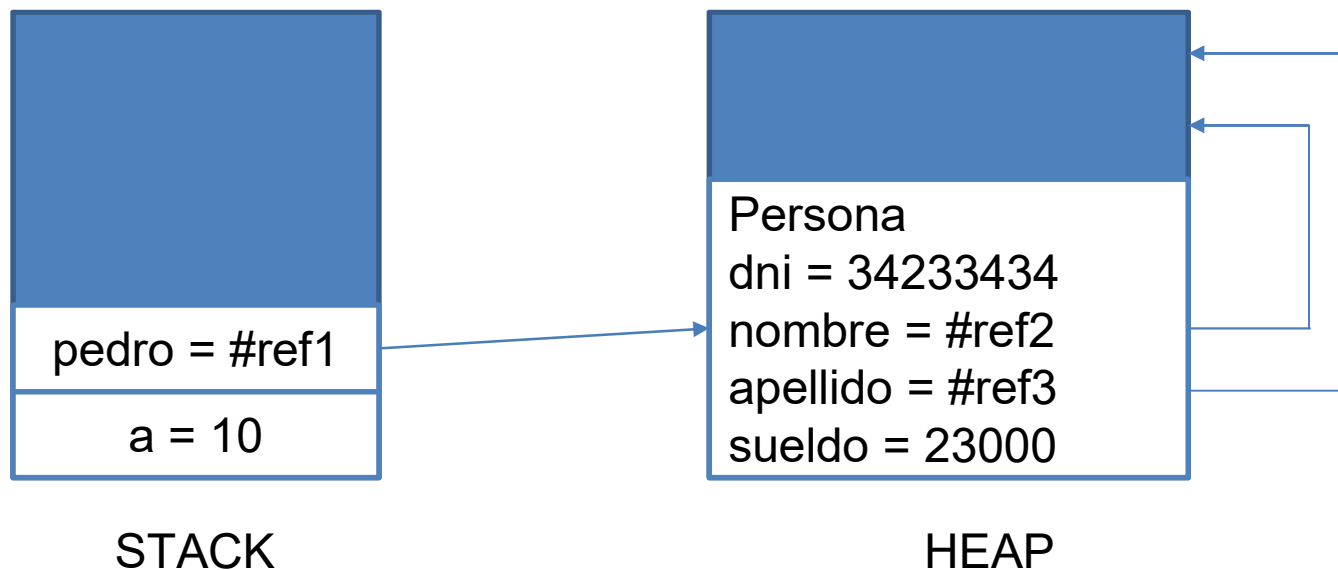
```
a *= 2 //a vale 40  
a /= 4 //a vale 10
```

# Referencia

- Las variable de tipo primitivo mantienen efectivamente el dato.
- Las variable de tipo clase mantienen una referencia a la instancia. Una referencia es la dirección de memoria donde se encuentra la instancia.

Suponga que se está ejecutando el siguiente código:

```
Persona pedro = new Persona(34_233_434, "Pedro", "Fernández", 23000);  
int a = 10;
```





# Referencia

¿Qué imprime este código por pantalla? ¿Porqué?

```
public static void main(String[] args) {  
    int a = 10;  
    int b = a;  
  
    System.out.println("La variable a vale: " + a);  
    System.out.println("La variable b vale: " + b);  
  
    b++; //incrementa a b en 1. Ahora debería valer 11.  
  
    System.out.println("La variable a vale: " + a);  
    System.out.println("La variable b vale: " + b);  
}
```

# Referencia

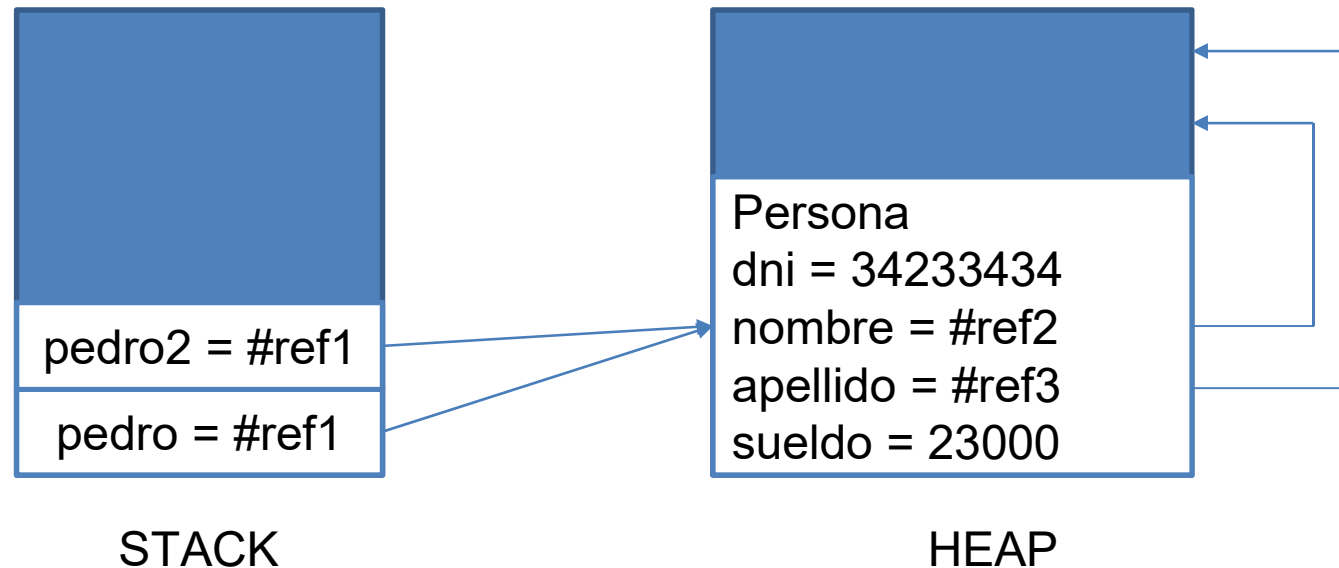
¿Qué imprime este código por pantalla? ¿Porqué?

```
public static void main(String[] args) {  
    Persona pedro = new Persona(34_233_434, "Pedro", "Fernández", 23000);  
    Persona pedro2 = pedro;  
  
    System.out.println("Apellido de pedro: " + pedro.getApellido());  
    System.out.println("Apellido de pedro2: " + pedro2.getApellido());  
  
    pedro2.setApellido("Diez");  
    System.out.println("Apellido de pedro: " + pedro.getApellido());  
    System.out.println("Apellido de pedro2: " + pedro2.getApellido());  
}
```

# Referencia

¿Qué imprime este código por pantalla? ¿Porqué?

```
public static void main(String[] args) {  
    Persona pedro = new Persona(34_233_434, "Pedro", "Fernández", 23000);  
    Persona pedro2 = pedro;  
  
    System.out.println("Apellido de pedro: " + pedro.getApellido());  
    System.out.println("Apellido de pedro2: " + pedro2.getApellido());  
  
    pedro2.setApellido("Diez");  
    System.out.println("Apellido de pedro: " + pedro.getApellido());  
    System.out.println("Apellido de pedro2: " + pedro2.getApellido());  
}
```





Ministerio de  
Educación y Deportes  
Presidencia de la Nación



Ministerio de Producción  
Presidencia de la Nación

