

# Taller 0 – Recordatorio Java

El propósito de este taller es recordar el funcionamiento general de una aplicación en java, su estructura y algunas herramientas para ejecutar aplicaciones Java.

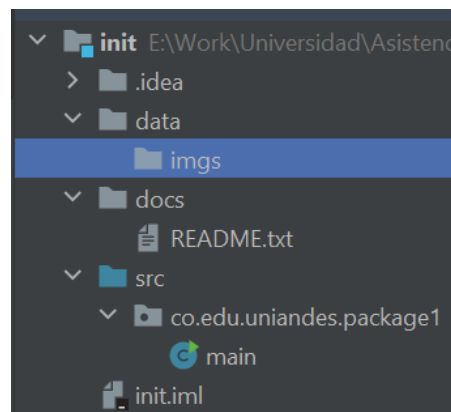
## 1 Generalidades

### Java

Java es un lenguaje de programación **orientado a objetos** y **tipado**, el cual se caracteriza por ser un lenguaje fácil de entender (de ahí que actualmente se siga usando para introducir a la programación), neutral a la arquitectura (32 bits o 64 bits) y es independiente de la plataforma gracias a la **Java Virtual Machine (JVM)**. Adicionalmente, su robusta estructura permite el uso de **múltiples hilos de ejecución**.

### Estructura de un proyecto

Los proyectos de java se dividen en **paquetes**, los cuales se almacenan en los directorios principales. En general, se tiene el directorio principal *src*, *data* y docs, donde se almacenarán los distintos paquetes que contendrán la aplicación, el contenido usado por el programa y la documentación del proyecto respectivamente.



La estructura interna de estos directorios se decide de acuerdo con el proyecto. Sin embargo, el estándar es nombrar los paquetes en minúscula y como prefijo el reverso del dominio de la organización (como se puede ver el caso de la universidad en el ejemplo).

Así mismo, para nombramiento de variables se utiliza el estándar **camelCase**.

## 2 Programando en java

Para programar en java, es necesario tener una serie de conceptos básicos claros. Para eso se cubrirá brevemente los siguientes conceptos: clases, objetos, variables, métodos y

visibilidad (de objetos y variables). A partir de estos conceptos se tiene la base de la estructura del lenguaje.

**Caso:** Para entender mejor los conceptos, se propone un caso de estudio simple, se desea construir un programa para administrar las notas de una materia de la universidad. En este se tendrán a los estudiantes y profesores, los estudiantes entregan trabajos o evaluaciones y todos pueden conocer las calificaciones, sin embargo, solamente el profesor puede modificar estas calificaciones.

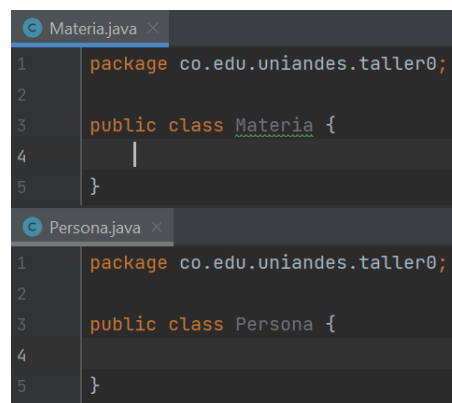
Tenga en cuenta el caso mientras se avanza en el taller.

## Objetos

Las clases de java se usan para modelar elementos de la realidad dentro del programa. Esos elementos tienen características y comportamientos, los cuales son modelados en una clase por medio de *atributos* y *métodos* respectivamente. Durante la ejecución de un programa, es posible crear instancias de una clase, es decir objetos, que tienen asociados estados particulares. Piense en el caso de estudio, ¿Cuáles serían las clases apropiadas para modelar el problema?

Un modelo puede ser tan sencillo o complejo como requiera la aplicación, para este caso solo necesitaríamos modelar materia y persona.

Para definir “Materia” y “Persona” cree una clase nueva con la definición que se muestra en los siguientes fragmentos de código (note que los nombres de la clase comienzan en mayúscula):



```
Materia.java
1 package co.edu.uniandes.taller0;
2
3 public class Materia {
4     |
5 }

Persona.java
1 package co.edu.uniandes.taller0;
2
3 public class Persona {
4
5 }
```

Antes de continuar con el modelaje, es importante recordar que los objetos de java cumplen con el concepto de **encapsulación de datos**. Este concepto restringe el acceso de un objeto a los atributos y métodos de otro objeto y se conoce como la **privacidad de un objeto o un atributo**. Java permite definir tres tipos de privacidad *public*, *private* y *protected* (en este curso solo usaremos los dos primeros).

- Algo declarado como público puede ser consultado y modificado por cualquier otro objeto.
- Algo privado solamente puede ser consultado y modificado por el objeto que lo contiene.

En el caso de estudio, se define la privacidad de ambas clases como pública para poder utilizarlos en todo el programa. Ahora debemos profundizar un poco y definir los atributos y métodos de estas clases.

## Variables y constantes

Al ser un lenguaje tipado, en java definimos una **variable** por medio de la siguiente estructura:

*[privacidad] tipoVariable nombreVariable;*

La privacidad es uno de los valores que se mencionaban en la *encapsulación de datos*. En cuanto a los tipos, en java se tiene la distinción de los tipos de datos primitivos (datos que son básicos y definidos en java en sí) o pueden ser otros objetos definidos por el programador o una librería (como en nuestro caso Materia y Persona). A continuación, una pequeña lista de los tipos primitivos:

Representa	Tipo en java
<b>Enteros</b>	int / short
<b>Decimales</b>	double / float
<b>Enteros de gran tamaño</b>	long
<b>Bytes</b>	byte
<b>Valor lógico</b>	boolean
<b>Un carácter</b>	char

También podemos encontrar algunas clases útiles definidas en java que se basan en los tipos primitivos. Entre ellos:

Representa	Tipo en java
<b>Cadena de caracteres</b>	String
<b>Fechas</b>	Date / Time / Calendar
<b>Numéricos</b>	Integer / Double / Long
<b>Caracteres</b>	Char

Finalmente, para los nombres de las variables y métodos, se sigue la convención de CamelCase, donde las variables siempre inician en minúscula y en cambio de palabra la primera letra va en mayúscula.

Otro tipo derivado útil son las matrices y los arreglos, que son listas. Sin embargo, se hace distinción entre las listas que son dinámicas (listas) y las que son de tamaño fijo (arreglos). Por ejemplo, una **lista de tamaño variable** se declara como:

*privacidad ArrayList < tipo > nombreLista = new ArrayList < tipo > ();*

Mientras que para un **arreglo de tamaño fijo**, debemos establecer el tamaño (cantidad de elementos) que va a contener. Por ejemplo:

*privacidad tipo[] nombreLista = new tipo[tamaño];*

Finalmente, para declarar **una constante** (un valor que no va a cambiar durante la ejecución del programa) se agrega entre la visibilidad y el tipo “*final static*”. Por un lado, “*final*” indica que el valor asignado a esa variable no va a poder cambiar nunca durante la ejecución, y “*static*” indica que el valor de esta variable será el mismo para todas las instancias de la clase. Es la combinación de estas palabras designadas las que permiten definir una constante. En cuanto al nombre, la convención es definir las constantes con mayúsculas y “\_” entre cambios de palabra. Por ejemplo:

*privacidad final static tipo NOMBRE\_CTE = valor;*

Para entender mejor, defina los siguientes atributos y constantes en cada una de las clases creadas anteriormente ¿Faltaría incluir algo para lograr cumplir el enunciado?

```
Materia.java
1 package co.edu.uniandes.taller0;
2
3 public class Materia {
4
5     public final static int MAX_ESTUDIANTES = 30;
6
7     private double[] calificaciones;
8
9     private Persona profesor;
10
11     private Persona[] estudiantes;
12
13 }
```

```
Persona.java
1 package co.edu.uniandes.taller0;
2
3 public class Persona {
4
5     private String codigo;
6
7     private String nombre;
```

Note que en este ejemplo la constante es de uso público. Por otro lado, los atributos son declarados como privados ya que solo queremos que puedan ser modificados por los objetos que los contienen.

## Métodos y operadores

Los métodos representan las acciones que puede realizar los objetos de las clases que modelamos. Lo recomendado es dividir los métodos de tal forma que podamos *reutilizar* su lógica (para evitar duplicar código) y que realicen funciones específicas de la lógica (uno para modificar, uno para exponer un atributo, etc.).

Para declarar un método, debemos tener en cuenta qué valores de entrada necesita, y si al final este arroja un valor. Estos valores se conocen como parámetros y retorno, respectivamente. Al igual que las variables, se debe especificar los tipos. Un método se declara de la siguiente manera:

```
privacidad tipoRetorno nombreMetodo(tipoP nombreParam, tipoP nombreParam2){}
```

Si la función no retorna ningún valor (por ejemplo, si solo modifica algo dentro del objeto) se puede especificar “void” como tipo de retorno. Por otro lado, si un método no recibe parámetros el espacio entre los paréntesis se deja vacío.

Al igual que los objetos, los métodos de java también cumplen con el principio de encapsulamiento, por lo cual, las variables que se declaran dentro de un método solamente existen dentro de este y no es posible acceder a su valor después de terminada la ejecución del método. A este tipo de variables se les conoce como variables temporales y al existir solamente dentro del método, no es necesario definir su privacidad.

Finalmente, en java se hace la distinción entre dos tipos de métodos. El método constructor es el método que permite generar una instancia de una clase (es decir, asignar valores a un objeto específico) y se caracteriza porque **no se le especifica tipo y tiene el mismo nombre que la clase**. El resto de los métodos se definen como se mencionó anteriormente y siempre se debe especificar el tipo de retorno.

Sin embargo, existe un método único que señala el punto de inicio de una aplicación Java conocido como “main”; cuando se ejecuta una aplicación Java la ejecución siempre inicia en este método. En general, en el main se asigna el estado inicial de la aplicación asignando valores de inicio y creando objetos.

La siguiente figura presenta los métodos definidos para cada clase del caso de estudio. Adicione el código indicado en las clases correspondientes.

```
9      // Constructor
10     public Persona(String pCodigo, String pNombre){
11         codigo = pCodigo;
12         nombre = pNombre;
13     }
14
15     // Getters
16     public String getCodigo(){
17         return codigo;
18     }
19
20     public String getNombre(){
21         return nombre;
22     }
```

```

public Materia(String pCodigoProfesor, String pNombreProfesor){
    calificaciones = new double[MAX_ESTUDIANTES];
    estudiantes = new Persona[MAX_ESTUDIANTES];
    profesor = new Persona(pCodigoProfesor, pNombreProfesor);
}

public double[] getCalificaciones() {
    return calificaciones;
}

public Persona getProfesor(){
    return profesor;
}

public Persona[] getEstudiantes() {
    return estudiantes;
}

public static void main(String[] args) {
    Materia infracomp = new Materia("123456", "Sandra Rueda");
}

```

En estos ejemplos, puede ver que siempre se tiene el método constructor en primer lugar, seguido por los métodos de la clase. Finalmente, la clase “Materia” contiene el método main, donde empieza la ejecución de la aplicación.

## Objetos parte 2

Habrás notado que, al utilizar el método constructor de Materia, se utiliza la palabra *new*. En esta instrucción se crea una instancia de un objeto y se debe ejecutar siempre que se quiere crear un nuevo objeto. Esta palabra reservada separa un espacio en memoria el cual será dedicado a ese objeto, la JVM calculará y reservará en memoria el espacio necesario mientras exista al menos una referencia a este objeto.

Por otro lado, en la estructura del proyecto se mencionaron los paquetes. Estos paquetes buscan generar modularidad dentro de la aplicación y al igual que el resto de java cumplen con el concepto de encapsulamiento, por esta razón, cuando se crea un objeto de una clase particular y se quiere utilizar en otro paquete, se debe importar al principio del archivo, especificando el nombre del paquete. Las librerías se deben importar de la misma manera.

Los métodos definidos en una clase se pueden invocar desde otros métodos en la misma clase usando solo su nombre seguido de paréntesis. Pero, si desde un objeto o1 se quiere invocar un método de un objeto o2, es necesario usar una referencia al objeto. Una vez tenemos la referencia a un objeto podemos utilizar sus métodos si de la siguiente manera:

*referencia.nombreMetodo(params ...)*

Si, un método devolviera un objeto, se puede acceder a los métodos de este objeto (siempre y cuando las condiciones de privacidad lo permitan). Si en el caso de estudio quisiéramos

acceder al nombre del profesor, debemos acceder primero al profesor y luego al nombre, de la siguiente manera:

```
public static void main(String[] args) {  
    Materia infracomp = new Materia("123456", "Sandra Rueda");  
    infracomp.getProfesor().getNombre();  
}
```

Estos llamados recursivos se pueden realizar sin límite, pero se recomienda no tener tantos métodos anidados de esta manera.

Adicionalmente, en java se tiene el concepto de **herencia**. En la herencia podemos hablar de padres e hijos, donde, una clase hija es una extensión de una clase padre, de tal forma que la hija hereda características del padre (métodos y atributos) siempre y cuando las condiciones de privacidad lo permitan. De esta forma, la clase hija podrá usar los atributos que se declaren en el padre.

Para indicar que una clase va a heredar de otra, se utiliza la palabra reservada *extends* indicando de quién se hereda:

*privacidad class ClaseHijo extends ClasePadre*

Ahora, para poder implementar que solamente un profesor pueda editar las notas, se debe modificar la clase “Persona” para que dos nuevas clases “Profesor” y “Estudiante” puedan heredar de esta. Elimine el método constructor y modifique la visibilidad de los atributos de la siguiente manera:

```
public class Persona {  
  
    protected String codigo;  
  
    protected String nombre;  
  
    // Getters  
    public String getCodigo() { return codigo; }  
  
    public String getNombre() { return nombre; }  
}
```

Ahora, se crearán dos nuevas clases de tipo “Profesor” y “Estudiante” la cuales heredan de la clase “Persona” y podrán utilizar las características de esta. Adicione el siguiente código en las clases correspondientes:

```
package co.edu.uniandes.taller0;

public class Profesor extends Persona{

    private double salario;

    private Materia materiaADictar;

    public Profesor(String pCodigo, String pNombre, Materia pMateriaADictar) {
        codigo = pCodigo;
        nombre = pNombre;
        salario = 1000000;
        materiaADictar = pMateriaADictar;
    }
}
```

```
package co.edu.uniandes.taller0;
import java.util.ArrayList;

public class Estudiante extends Persona{

    private double mesada;

    private ArrayList<Materia> materiasInscritas;

    public Estudiante(String pCodigo, String pNombre) {
        codigo = pCodigo;
        nombre = pNombre;
        materiasInscritas = new ArrayList<>();
        mesada = 300000;
    }
}
```

Finalmente, cambiaremos el tipo de los objetos “profesor” y “estudiantes” en la clase Materia y en un nuevo método, verificaremos que quien llama al método para editar las calificaciones sea un profesor y este pueda editar las notas con las nuevas notas que desee, si no, se informará que no se pueden editar las notas.

```
private Profesor profesor;

private Estudiante[] estudiantes;

public Materia(String pCodigoProfesor, String pNombreProfesor){
    calificaciones = new double[MAX_ESTUDIANTES];
    estudiantes = new Estudiante[MAX_ESTUDIANTES];
    profesor = new Profesor(pCodigoProfesor, pNombreProfesor, this);
}

public void setNotas(Persona pEditor, double[] pNuevasCalificaciones){
    if(pEditor instanceof Profesor)
        calificaciones = pNuevasCalificaciones;
    else
        System.out.println("No puedes editar las notas");
}
```



Además, se deben actualizar los tipos de retorno de los métodos en Materia para concordar con los nuevos tipos:

```
public Profesor getProfesor() { return profesor; }

public Estudiante[] getEstudiantes() { return estudiantes; }
```

Por el lado del profesor, al momento de actualizar las notas tendría que enviar una instancia de sí mismo (así el objeto que se invoca puede comprobar que la acción solicita es invocada por un profesor) y las nuevas notas a la materia que le interese, en este caso, solo tiene una materia y quiere subir la calificación de un estudiante y poder modificarla.

Para realizarlo, agregue los siguientes métodos en la clase Profesor:

```
public void publicarNotas(){
    double[] nuevasNotas = new double[Materia.MAX_ESTUDIANTES];
    nuevasNotas[0] = 5.0;
    materiaADictar .setNotas(this, nuevasNotas);
}

public void actualizarNota(int pPos, double pNota){
    double[] notas = materiaADictar.getCalificaciones();
    notas[pPos] = pNota;
    materiaADictar.setNotas(this, notas);
}
```

Para que un usuario pueda correr el programa vamos a modificar el flujo del código para que cuando inicie la ejecución en el método *main* el profesor ingrese la nueva nota del estudiante y esta se guarde en las calificaciones de la materia. El siguiente código permite pedir al usuario que ingrese la nueva nota y al final, escribirá el valor de la nota guardada en consola.

```
public static void main(String[] args) {
    System.out.println("Iniciando programa");
    Materia infracomp = new Materia("123456", "Sandra Rueda");
    infracomp.getProfesor().publicarNotas();
    System.out.println("Ingrese la nueva nota");
    Scanner sc = new Scanner(System.in);
    double nuevaNota = Double.parseDouble(sc.nextLine());
    infracomp.getProfesor().actualizarNota(0, nuevaNota);
    System.out.println("La nueva nota es: " + infracomp.getCalificaciones()[0]);
}
```

### 3 Ejecutando un programa

Para ejecutar un programa en java es necesario tener en el dispositivo la JVM, a lo largo del tiempo se han lanzado varias versiones (hasta Java 18 en versión LTS, o 20 en acceso anticipado), pero la recomendada para desarrollo y en general por su estabilidad es la versión 8.

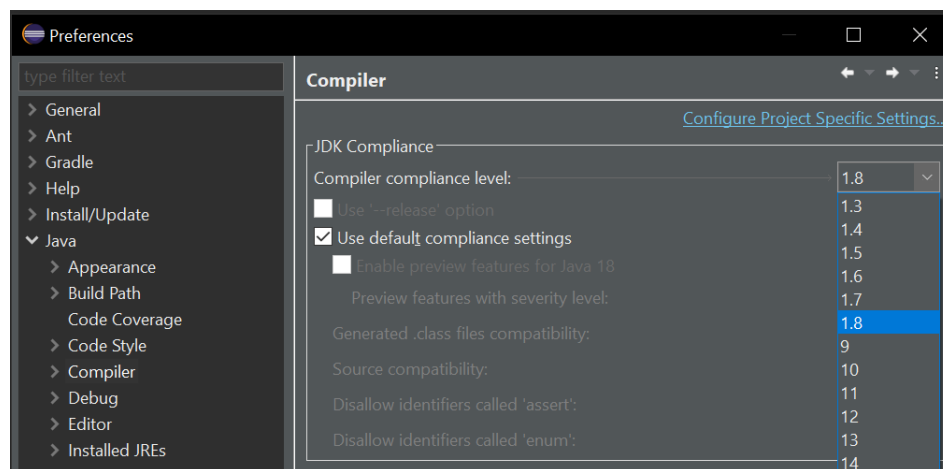
Ahora bien, para poder desarrollar en java es necesario compilar el código utilizando el paquete para desarrolladores de java o JDK por sus siglas en inglés (*Java SE Development Kit*). Y para ejecutar la aplicación se utiliza el ambiente de ejecución de Java o JRE por sus siglas en inglés (*Java SE Runtime Environment*). Ambos están disponibles de manera gratuita para su descarga desde el [sitio web de Oracle](#).

Una vez instalados en el computador, se pueden utilizar IDEs para ver los proyectos, compilar y ejecutar el código. En este caso se explicará cómo hacerlo desde eclipse y desde IntelliJ. Puede utilizar el de su preferencia, al final de la configuración, se explicarán algunas cosas extra que podrá realizar desde cualquiera de los IDEs.

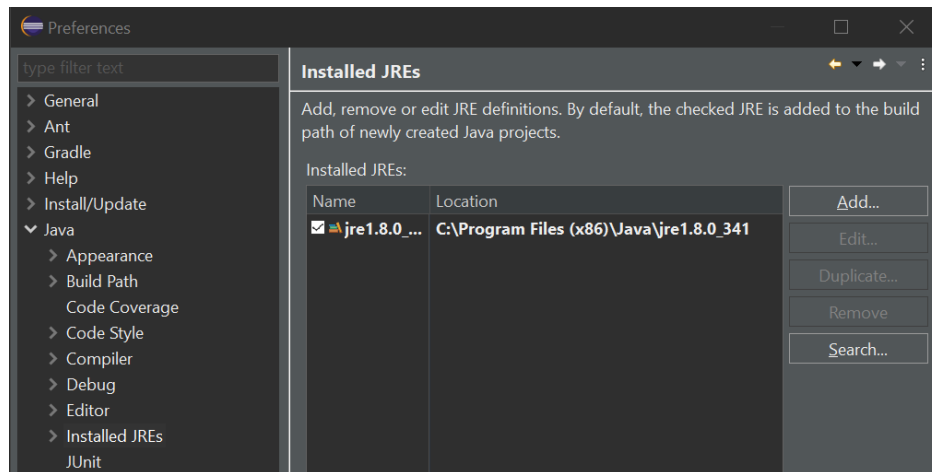
### Eclipse

Eclipse es un IDE mantenido por la comunidad con distintos sabores de acuerdo con las necesidades del desarrollador (como el tipo de aplicación a desarrollar). Puede utilizar la versión que descargo en cursos anteriores, o puede descargar la última desde el [sitio oficial de eclipse](#).

Una vez termine la descarga e instalación, no olvide **configurar el JRE y el JDK** a utilizar. Para configurar el JDK diríjase a Window->Preferences->Java->Compiler. Allí, seleccione la versión 1.8

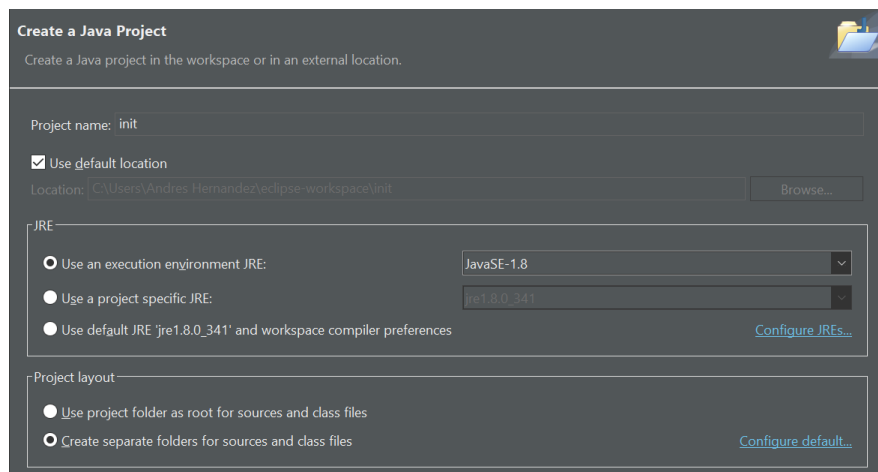


Para configurar el JRE diríjase a Window->Preferences->Java->Installed JREs y seleccione la versión 1.8. Si esta no aparece disponible, puede agregarla manualmente en el botón “Add” ubicando la ruta de instalación general del JRE.



Puede tener varias versiones de Java instaladas en su computador, sin embargo, las aplicaciones **deben ser compiladas y ejecutadas en la misma versión**. Si las versiones no coinciden, su programa no ejecutará y Java arrojará una excepción. **Para el curso utilizaremos la JVM 1.8**, por lo que, si decide utilizar otra versión, recuerde comprobar que el programa compile y ejecute para Java 1.8.

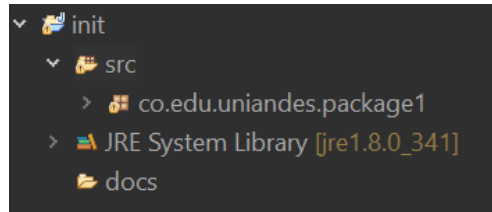
Una vez configurado el ambiente, podemos crear la primera aplicación. Cree un nuevo proyecto de java (File->new->Java Project), especifique el nombre del programa, donde lo quiere guardar y verifique que este la versión de java configurada.



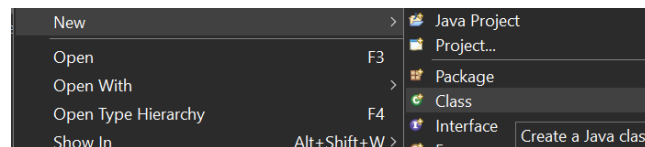
Cree un nuevo paquete con el dominio reverso de la universidad y el nombre package1.



Debería verlo así:



Ahora, sobre este paquete, cree un nuevo archivo java, con el nombre main.

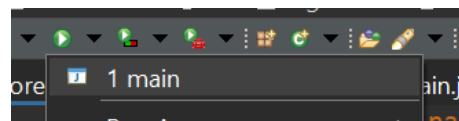
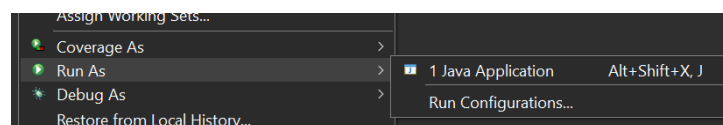


El archivo tendrá el siguiente contenido:

```
1 package co.edu.uniandes.package1;
2
3 public class main {
4
5     public static void main(String[] args) {
6         System.out.println("Hello world!");
7     }
8 }
9 }
```

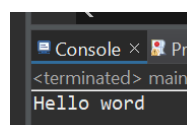
En orden, se indica el paquete al que pertenece la clase, se declara la misma (en este caso declaramos la clase principal *main*) y ponemos el método principal de la aplicación. Por defecto Java **siempre ejecutara primero el método main**. Recuerde al crear sus aplicaciones definir el estado inicial de la aplicación desde allí.

Para ejecutar el programa, puede dar click derecho al archivo ->Run As-> Java Application o desde el botón de la barra de herramientas para desarrollador. También puede realizar la ejecución predeterminada con ctr+F11.



En caso de que no le aparezca ninguna aplicación, verifique que haya declarado correctamente el método principal.

Al terminar la ejecución, debería aparecer en consola el mensaje "Hello world".

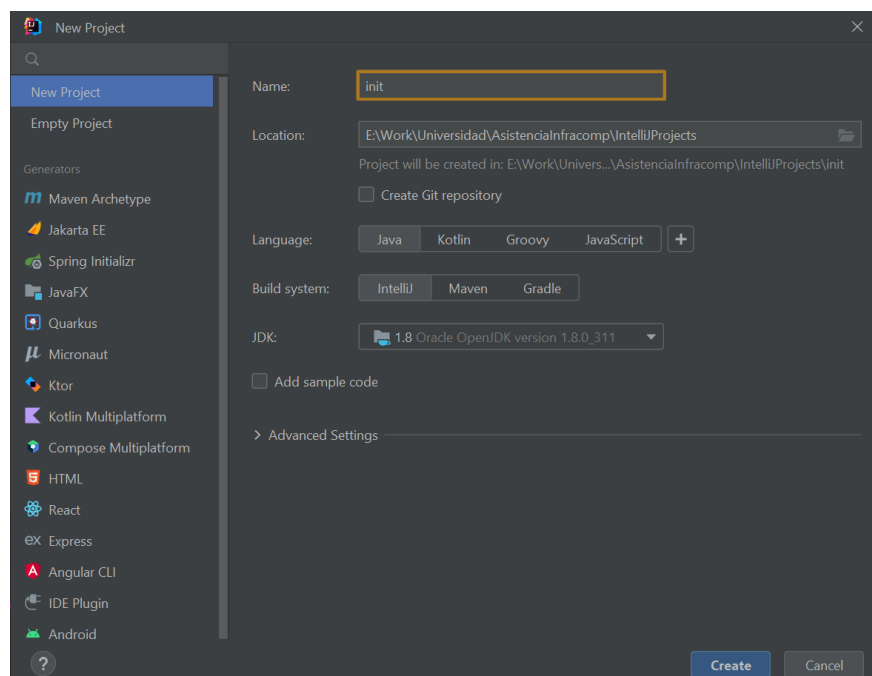


## IntelliJ IDEA

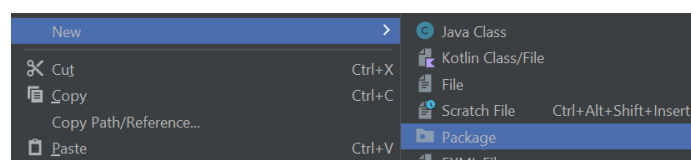
Este IDE fue desarrollado por JetBrains (la misma empresa que desarrollo PyCharm) con una interfaz más moderna con capacidad de soportar múltiples tipos de aplicación, compilación y cambiar la interfaz según se necesite. En nuestro caso, se ejecutarán aplicaciones Java, pero también se podrían ejecutar servidores con Maven o aplicaciones móviles en Java o Kotlin. En el caso de Eclipse, sería necesario instalar otro sabor de eclipse especializado en esto.

La universidad cuenta actualmente con la licencia de estudiantes de JetBrains, por lo que puede utilizar la versión completa del programa creando su cuenta con su correo institucional e iniciando sesión una vez instale el programa. La descarga la puede realizar [aquí](#).

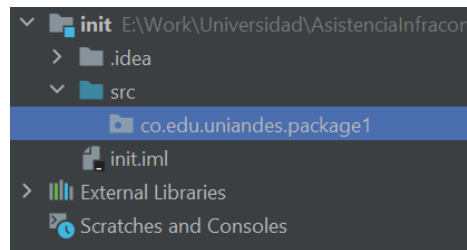
Una vez inicie sesión, el programa reconocerá automáticamente las versiones de java en su computadora y podrá crear un nuevo proyecto. Indique la ruta que desea y seleccione el Java e IntelliJ. Verifique que el JDK seleccionado sea el 1.8 y continúe. (¡También puede crear al mismo tiempo un repositorio de Git para trabajar con su compañero!)



Cuando termine la creación del proyecto, cree un nuevo paquete con el dominio reverso de la universidad y el nombre package1.



Debería verlo así:



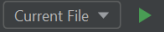
Ahora, sobre este paquete, cree un nuevo archivo java, con el nombre main.

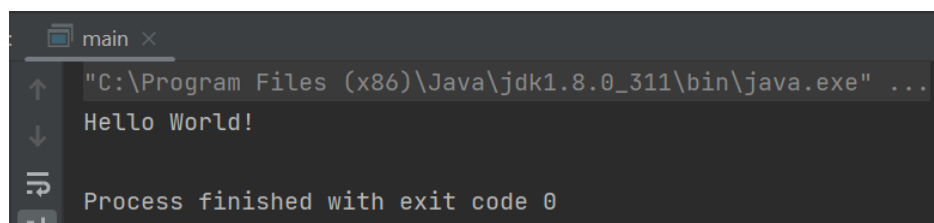


El archivo tendrá el siguiente contenido:

```
main.java x
1 package co.edu.uniandes.package1;
2
3 public class main {
4     public static void main(String[] args) {
5         System.out.println("Hello World!");
6     }
7 }
8
```

En orden, se indica el paquete al que pertenece la clase, se declara la misma (en este caso declaramos la clase principal *main*) y ponemos el método principal de la aplicación. Por defecto Java **siempre ejecutara primero el método main**. Recuerde al crear sus aplicaciones definir el estado inicial de la aplicación desde allí.

Para ejecutar el programa, puede dar click en el botón run en la barra de ejecución  o puede presionar shift + F10. Si es la primera vez que lo ejecuta, espere un poco mientras se compila el programa y se genera el ejecutable. Al terminar la ejecución, debería aparecer en consola el mensaje "Hello world".



## 4 Conclusiones

Hay más clases definidas en java que permiten modelar estructuras más complejas, si desea profundizar su conocimiento existen múltiples fuentes en línea para aprender a programar en java como [W3Schools](#), [GeeksForGeeks](#) o [JavaTPoint](#). También puede consultar video tutoriales o consultar con los profesores sobre las dudas que tenga.

## Historial de revisiones

### Registro de cambios

<b>Fecha</b>	<b>Autor(es)</b>	<b>Versión</b>	<b>Referencia de Cambios</b>
01/08/2022	Andrés Hernández	0.1	Versión Inicial

### Revisores

<b>Nombre</b>	<b>Cargo</b>	<b>Institución</b>
Sandra Rueda	Profesor Asociado Departamento de Ingeniería de Sistemas y Computación	Universidad de los Andes
Harold Castro	Profesor Titular Departamento de Ingeniería de Sistemas y Computación	Universidad de los Andes