

CURSO DESARROLLO DE APLICACIONES MÓVILES ANDROID TRAINEE

Módulo 1.

Programación básica en java (5 unidades)

Temario

Unidad 1

a) Algoritmos

b) Estructuras de control condicional

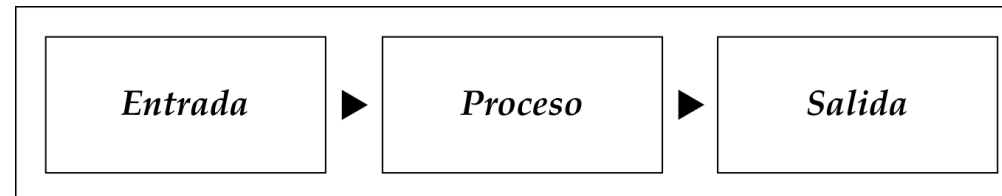
c) Diagramación de algoritmos

Unidad 1

Algoritmos

Un algoritmo se puede definir como una secuencia de instrucciones que representan un modelo de solución para determinado tipo de problemas. O bien como un conjunto de instrucciones que realizadas en orden conducen a obtener la solución de un problema o situación.

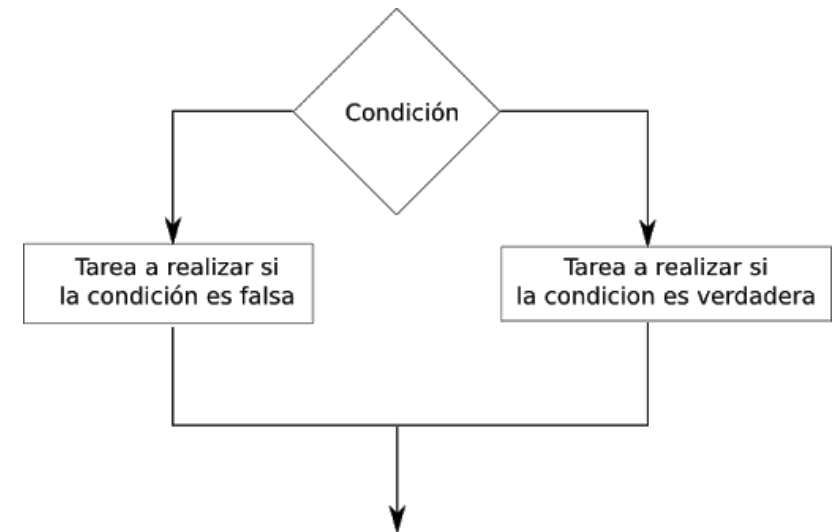
Todos los algoritmos se componen de 3 partes fundamentales que permiten su ejecución; Entrada, Proceso y Salida.



Unidad 1

Estructuras de control condicional

Las estructuras condicionales comparan una variable o dato contra otro valor o valores, para que en base al resultado de esta comparación, se siga un curso de acción dentro del algoritmo.




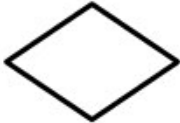



Unidad 1

Diagramación de algoritmos

La diagramación de algoritmos hace referencia principalmente a los diagramas de flujo los cuales proporcionan un medio para poder representar un proceso que posee una serie de pasos específicos claros para su realización estos pasos están en un algoritmo.

Los diagramas de flujo son contruidos mediante cuadros de diferentes maneras los cuales son símbolos que representan una función en particular, estos símbolos están interconectados para la indicación del proceso a realizar, en ellos se encuentran los pasos determinados en el algoritmo indicando la secuencia de distintas tareas.

Símbolo	Significado/Función
	Inicio/Fin/Conector
	Proceso
	Entrada/Salida
	Decisión
	Flujo de ejecución

Material complementario de la unidad

Link a video relacionado

Algoritmo - Fundamentos de la programación: <https://youtu.be/YFdXfehe2bo>

Link a lectura complementaria

¿Sabes que es un algoritmo?: <https://pandorafms.com/blog/es/que-es-un-algoritmo/>

Link a investigación relacionada

Algoritmos, aplicaciones y Big data, nuevos paradigmas en el proceso de comunicación y de enseñanza-aprendizaje del periodismo de datos: <http://www.scielo.org.pe/pdf/rcudep/v17n2/a13v17n2.pdf>

Temario

Unidad 2

- a) El Entorno Java para la programación
- b) Sentencias de control
- c) Creando aplicaciones de consola en Java

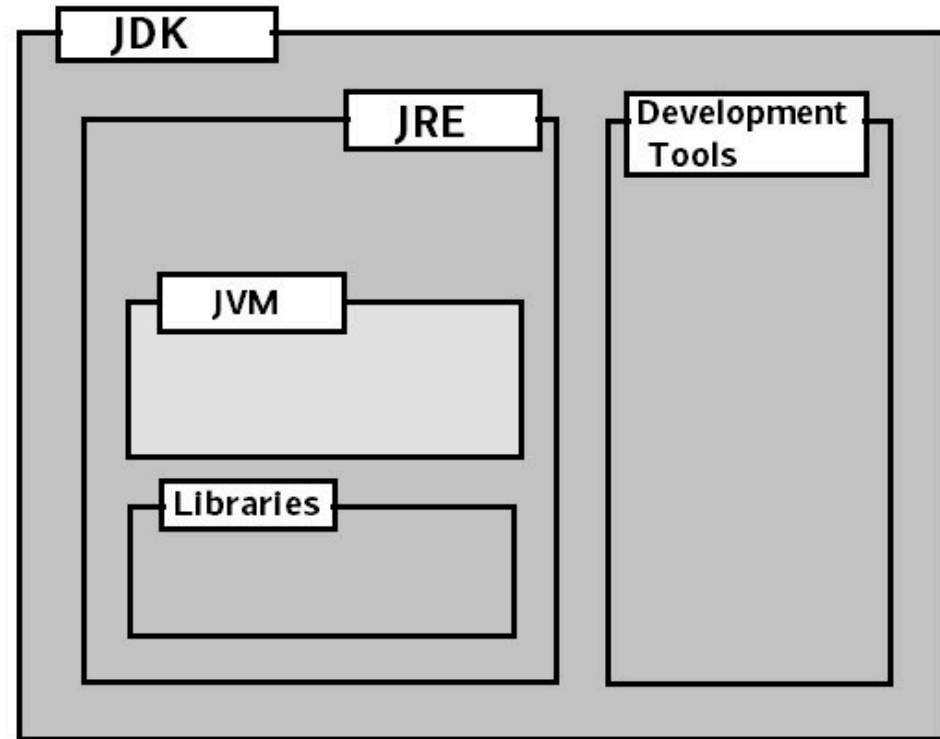
Unidad 2

El Entorno Java para la programación

Los entornos de desarrollo Java son aplicaciones que permiten al programador implementar las abstracciones del mundo real en un aplicación concreta mediante la introducción de secuencias de código con sus estructuras de programación.

Unidad 2

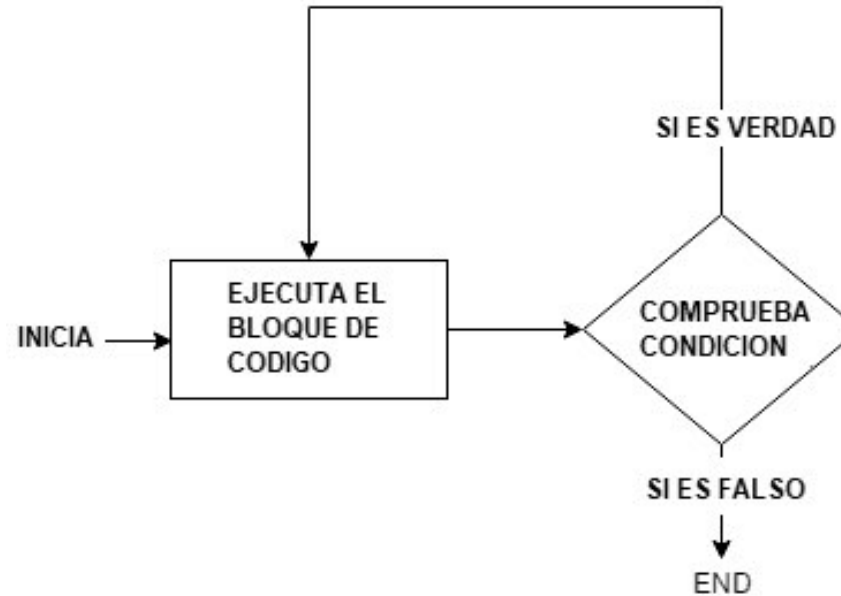
Los **entornos de desarrollo Java (IDE)** son herramientas que funcionan como un sistema integral que nos permite programar de forma mucho más sencilla que si solo empleásemos el JDK (*Java Development Kit*) de Java y un procesador de textos



Unidad 2

Sentencias de control

Un ciclo en Java o bucle en Java permite repetir una o varias instrucciones cuantas veces lo necesitemos o sea necesario



Unidad 2

si quisiéramos escribir los números del uno al cien no tendría sentido escribir cien líneas de código mostrando un número en cada una de estas, para eso y para varias cosas más, es útil un ciclo

Unidad 2

Existen diferentes tipos de ciclos o bucles en Java, cada uno tiene una utilidad para casos específicos y depende de nuestra habilidad y conocimientos poder determinar en qué momento o situación es bueno usar alguno de ellos. Tenemos entonces a nuestra disposición los siguientes tipos de ciclos en Java:

FOR

un ciclo for es una estructura iterativa para ejecutar un mismo segmento de código una cantidad de veces deseada; conociendo previamente un valor de inicio, un tamaño de paso y un valor final para el ciclo realizando finalmente el incremento en su contador para la siguiente iteración.

```
for (int i = valor inicial; i <= valor final; i = i ++)  
{  
    ....  
    Instrucciones....  
    ....  
}
```

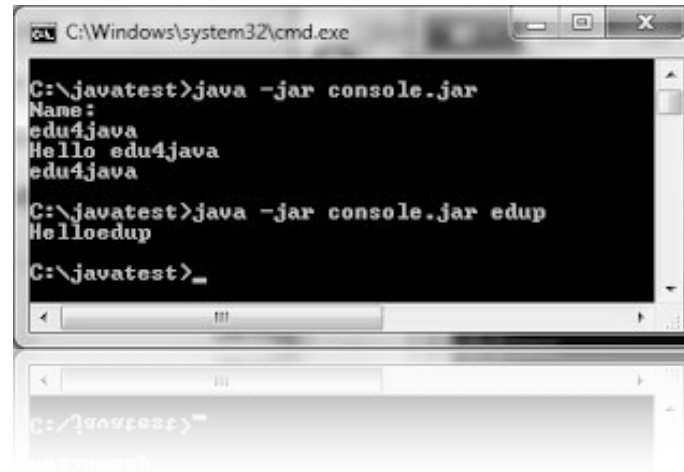
Unidad 2

<p>WHILE</p> <p>El bucle while presenta ciertas similitudes y ciertas diferencias con el bucle for. La repetición en este caso se produce no un número predeterminado de veces, sino mientras se cumpla una condición. Conceptualmente el esquema más habitual es el siguiente:</p>	<pre>Int i = 0; while (i <= valor final) { i++; }</pre>
<p>DO-WHILE</p> <p>El bucle do ... while es muy similar al bucle while. La diferencia radica en cuándo se evalúa la condición de salida del ciclo. En el bucle while esta evaluación se realiza antes de entrar al ciclo, lo que significa que el bucle puede no llegar ejecutarse. En cambio, en un bucle do ... while, la evaluación se hace después de la primera ejecución del ciclo, lo que significa que el bucle obligatoriamente se ejecuta al menos en una ocasión. A modo de ejercicio, escribe este código y comprueba los resultados que se obtienen con él.</p>	<pre>do { contador += 1; } while (contador < 10);</pre>

Unidad 2

Creando aplicaciones de consola en Java

Las aplicaciones de consola son aquellos programas en Java que se van a ejecutar en una ventana de comandos o de Shell, como vemos a continuación:



```
C:\Windows\system32\cmd.exe

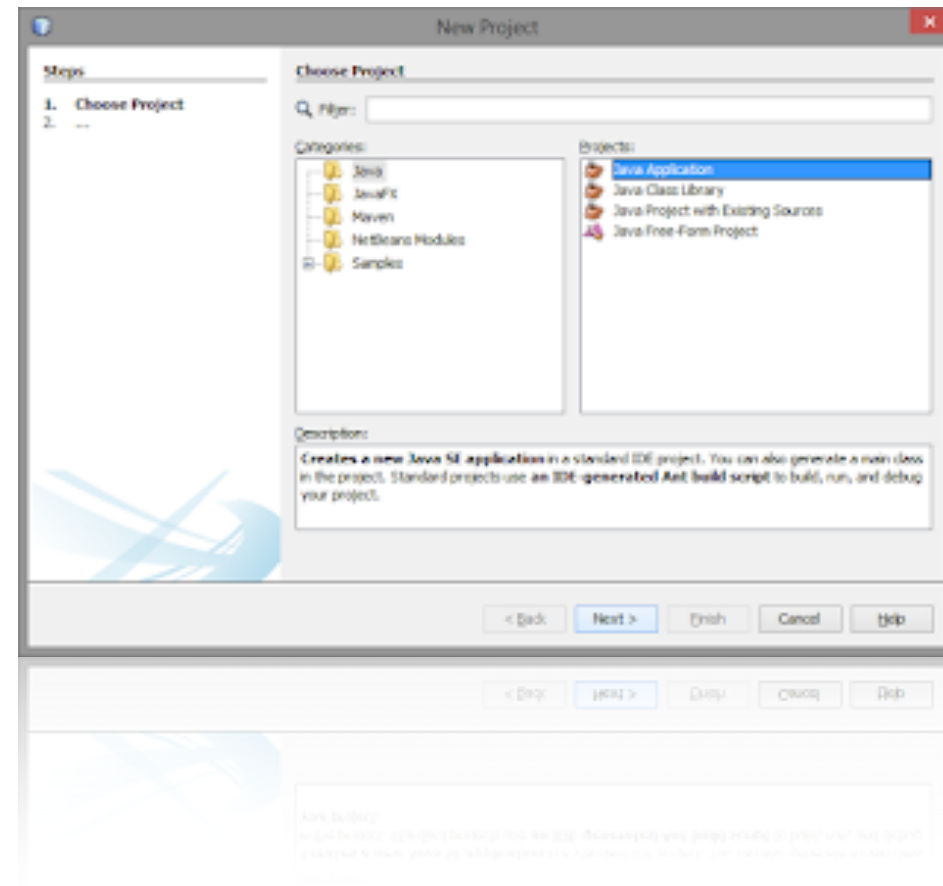
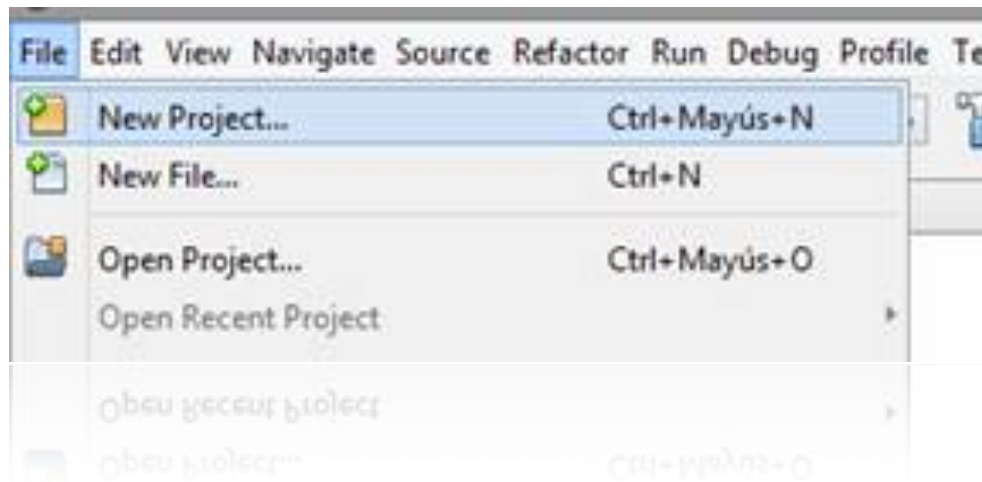
C:\javatest>java -jar console.jar
Name:
edu4java
Hello edu4java
edu4java

C:\javatest>java -jar console.jar edup
Helloedup

C:\javatest>
```

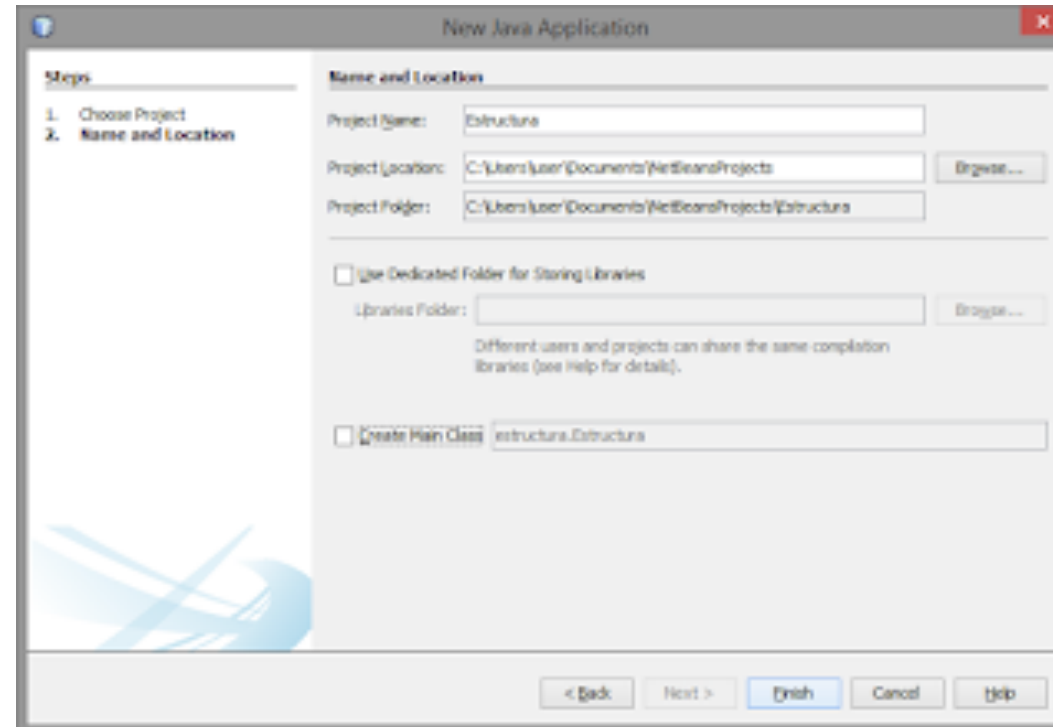
Unidad 2

Creamos un nuevo proyecto al que vamos a llamar Estructura:



Unidad 2

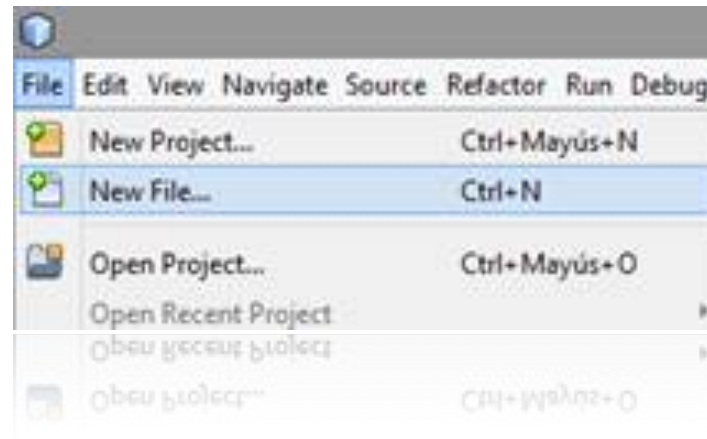
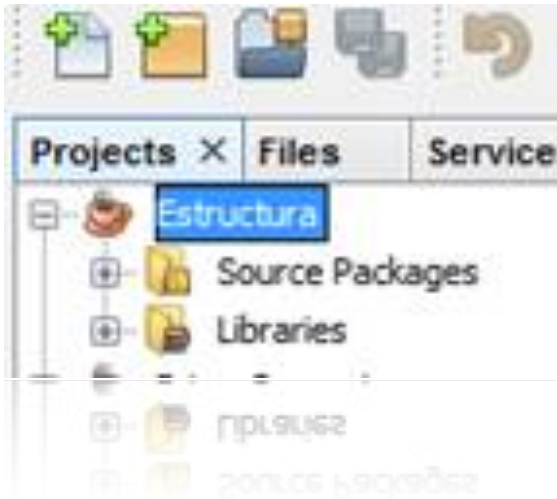
Desactivaremos Create Main Class:



Unidad 2

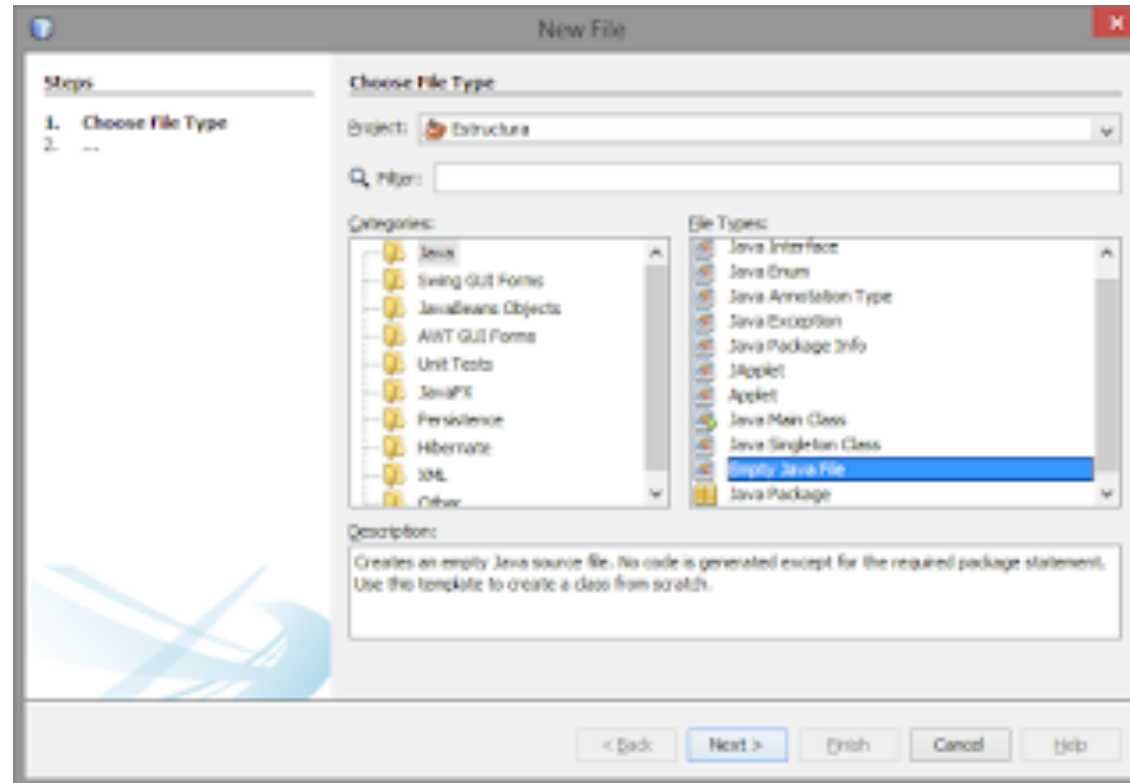
Hacemos clic en *Finish* y se nos crea un nuevo proyecto, pero sin ningún archivo:

nuevo fichero lo podemos hacer desde el menú File > New File... o haciendo clic en este icono



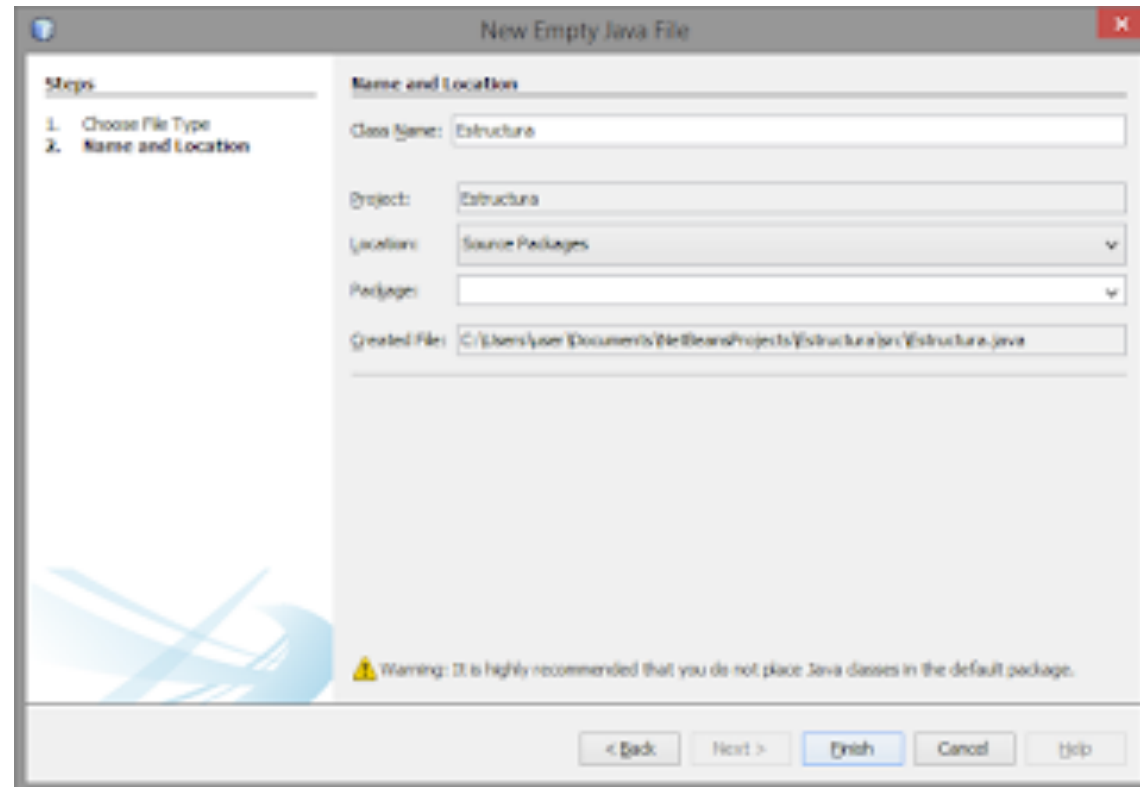
Unidad 2

Vamos a seleccionar como tipo de archivo, un archivo vacío Empty Java File:



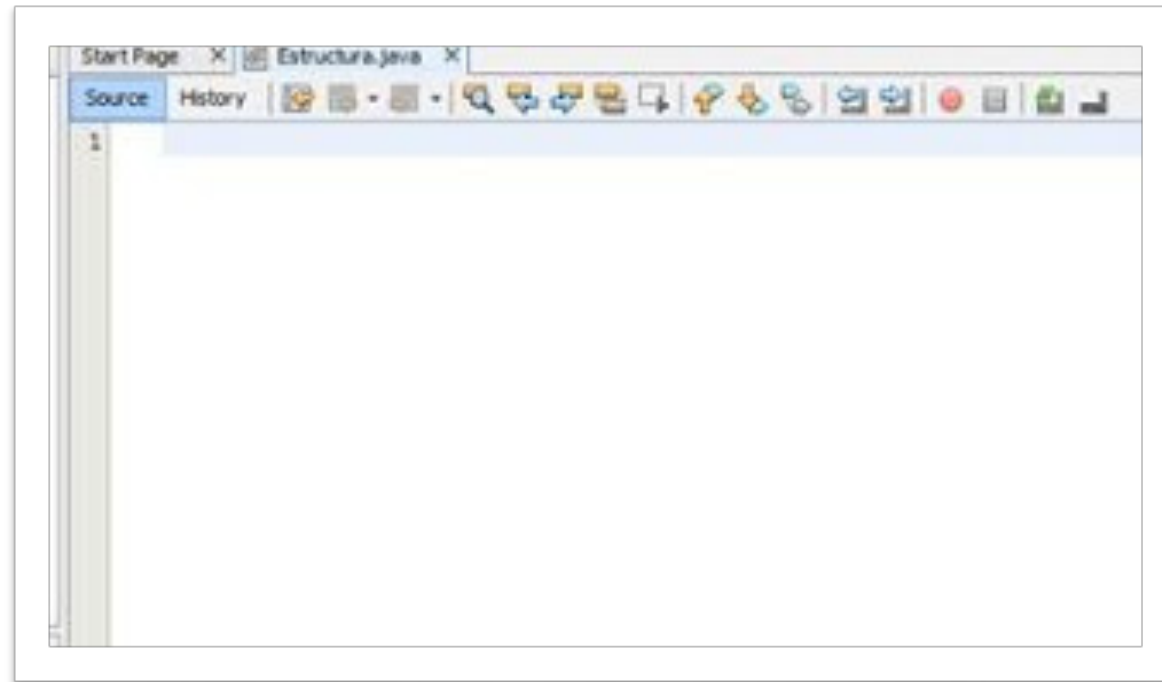
Unidad 2

Al fichero le vamos a llamar Estructura:



Unidad 2

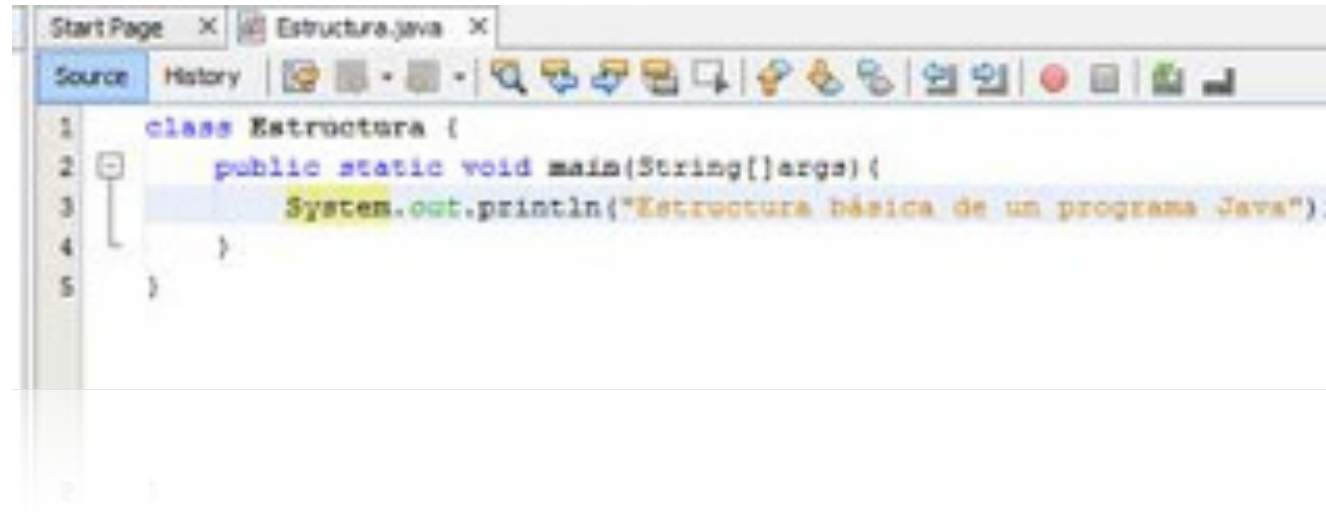
Y se nos ha creado un archivo vacío:



Unidad 2

Unidad 2

Vamos a empezar a escribir código en este archivo:



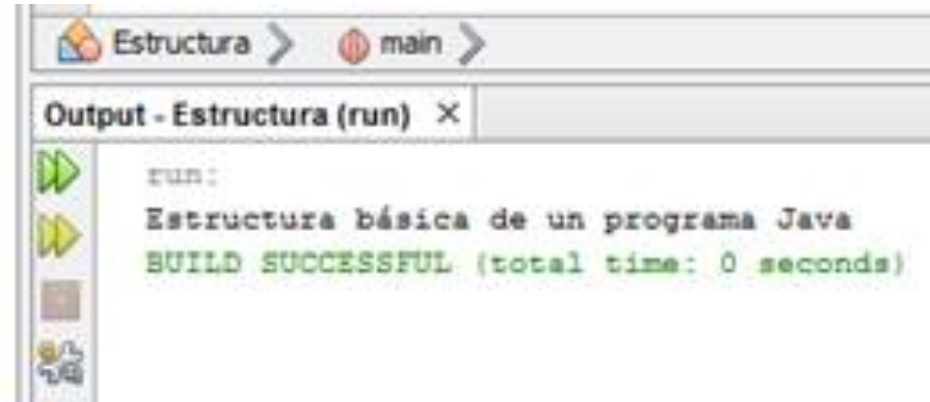
```
1 class Estructura {  
2     public static void main(String[] args) {  
3         System.out.println("Estructura básica de un programa Java");  
4     }  
5 }
```

Unidad 2

1. La declaración `class`, lo que hacemos es declarar una clase, el nombre de la clase debe coincidir con el nombre del archivo, en este caso `Estructura`.
2. La segunda línea es la declaración `main`, estamos indicando que la parte principal del programa empieza aquí, `main` es el punto de entrada de la mayoría de los programas, hay excepciones como por ejemplo los `applets` que son programas que se ejecutan como parte de una página Web, y los `servlets` que son programas que se ejecutan en un servidor.
3. Las llaves también forman parte de la estructura del programa, gracias a ellas creamos los bloques. En este ejemplo vemos que la llave de apertura de la línea uno tiene que ver con la llave de cierre de la línea cinco, forma un grupo, dentro de este grupo hay una llave de apertura y otra de cierre que forma un bloque dentro de ese grupo.
4. La función `System.out.println` nos va a imprimir en pantalla lo que le pasemos dentro de los paréntesis entre comillas.

Unidad 2

Si este programa lo ejecutamos nos aparece en pantalla lo que hay entre paréntesis en la función `System.out.println`:



Material complementario de la unidad

Link a video relacionado

Java y el entorno de desarrollo: <https://youtu.be/6mSBnIEzSms>

Link a lectura complementaria

<https://www.campusmvp.es/recursos/post/Los-mejores-entornos-de-desarrollo-para-Java.aspx>

<https://jarroba.com/instalar-bien-eclipse-un-ide-de-muchos/>

Temario

Unidad 3

a)El Paradigma de Orientación a Objeto

b)Clases y Objetos

c)Métodos Constructores

Unidad 3

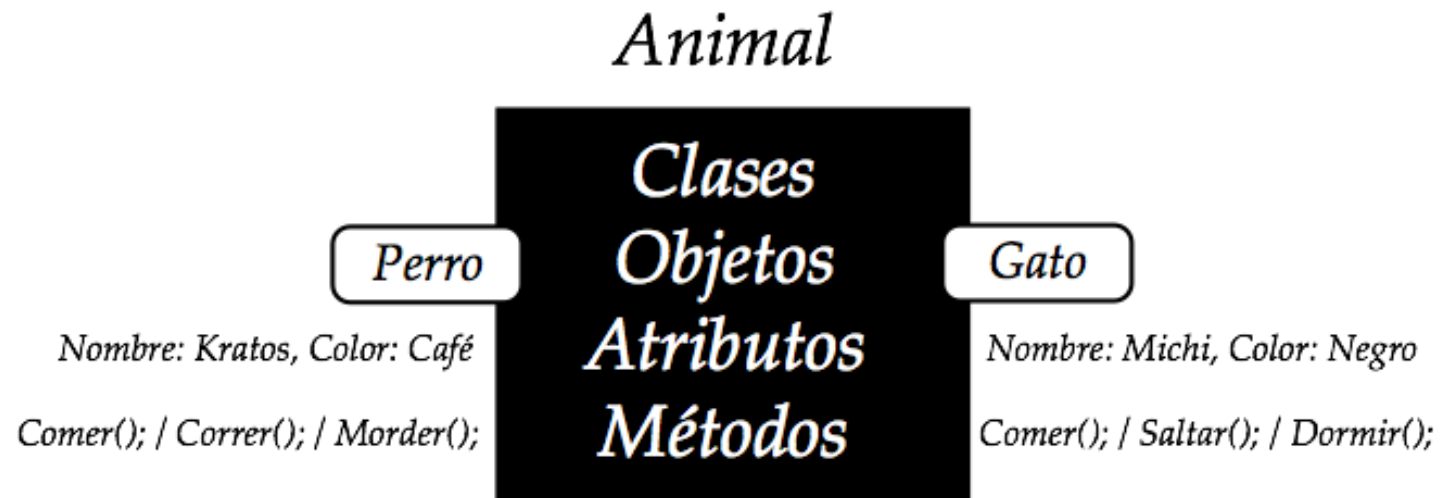
El Paradigma de Orientación a Objeto

Un paradigma de programación es una manera o estilo de programación de software. Existen diferentes formas de diseñar un lenguaje de programación y varios modos de trabajar para obtener los resultados que necesitan los programadores.

Unidad 3

Clases y Objetos

Una clase es una plantilla. Define de manera genérica cómo van a ser los objetos de un determinado tipo.



Unidad 3

Métodos Constructores

La solución en los lenguajes orientados a objetos es emplear los constructores. Un constructor es un método perteneciente a la clase que posee unas características especiales

- Se llama igual que la clase.
- No devuelve nada, ni siquiera void.
- Pueden existir varios, pero siguiendo las reglas de la sobrecarga de funciones.
- De entre los que existan, tan sólo uno se ejecutará al crear un objeto de la clase.

Material complementario de la unidad

Link a video relacionado

¿Qué es la programación orientada a objetos?

https://youtu.be/DlphYPc_HKk

Link a lectura complementaria

“un paradigma de programación que permite desarrollar aplicaciones complejas manteniendo un código más claro “

<https://desarrolloweb.com/articulos/499.php>

Temario

Unidad 4

a) Herencia y Polimorfismo

b) Principios básicos de diseño Orientado a Objetos

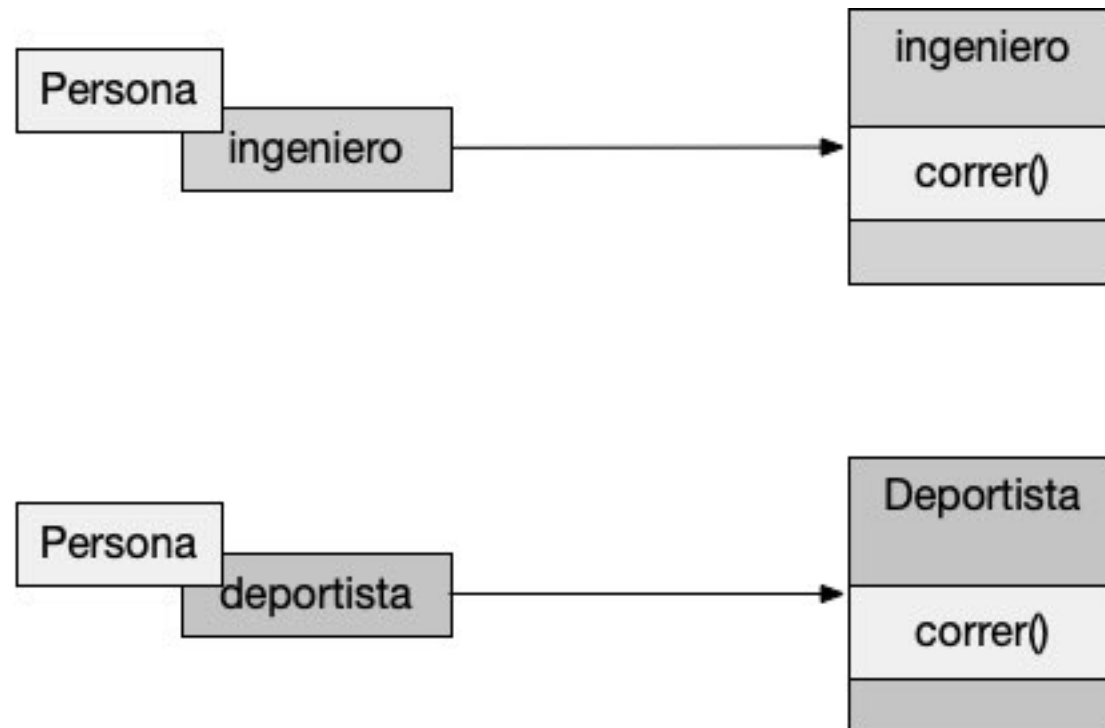
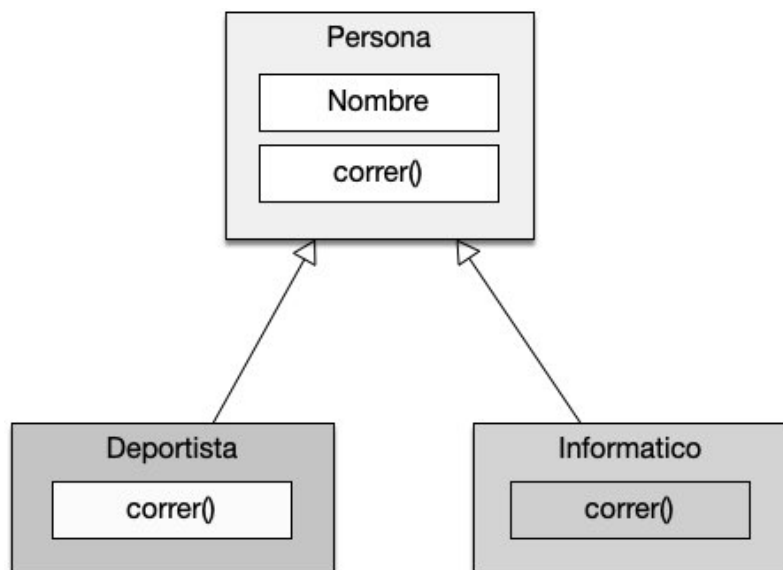
Unidad 4

Herencia y Polimorfismo

Herencia	Polimorfismo
<p>Las clases principales extienden atributos y comportamientos a las clases secundarias. A través de la definición en una clase de los atributos y comportamientos básicos, se pueden crear clases secundarias, ampliando así la funcionalidad de la clase principal y agregando atributos y comportamientos adicionales.</p>	<p>El polimorfismo consiste en diseñar objetos para compartir comportamientos, lo que nos permite procesar objetos de diferentes maneras. Es la capacidad de presentar la misma interfaz para diferentes formas subyacentes o tipos de datos.</p>



Unidad 4



El método correr de la clase Persona es un método abstracto y no tiene implementación . Por el contrario los métodos de la clases hijas tienen sobrecargado el método correr. El deportista correrá a 7 hm/hora y el informático a 2km/h .

Unidad 4

Principios básicos de diseño Orientado a Objetos

Introducción a los principios SOLID

SOLID es una palabra que debes conocer como uno de los fundamentos de la arquitectura y desarrollo de software.

SOLID es el acrónimo que acuñó Michael Feathers, basándose en los principios de la programación orientada a objetos que Robert C. Martin había recopilado en el año 2000 en su paper “Design Principles and Design Patterns”. Ocho años más tarde, el tío Bob siguió compendian- do consejos y buenas prácticas de desarrollo y se convirtió en el padre del código limpio con su célebre libro Clean Code.

Los 5 principios SOLID de diseño de aplicaciones de software son:

S – Single Responsibility Principle (SRP)

O – Open/Closed Principle (OCP)

L – Liskov Substitution Principle (LSP)

I – Interface Segregation Principle (ISP)

D – Dependency Inversion Principle (DIP)

S	RESPONSABILIDAD ÚNICA
O	ABIERTO/CERRADO
L	SUSTITUCIÓN DE LISKOV
I	SEGREGACIÓN DE INTERFACES
D	INVERSIÓN DE DEPENDENCIAS

Entre los objetivos de tener en cuenta estos 5 principios a la hora de escribir código encontramos:

- Crear un software eficaz: que cumpla con su cometido y que sea robusto y estable.
- Escribir un código limpio y flexible ante los cambios: que se pueda modificar fácilmente según necesidad, que sea reutilizable y mantenible.
- Permitir escalabilidad: que acepte ser ampliado con nuevas funcionalidades de manera ágil.

Material complementario de la unidad

Link a video relacionado

Principios SOLID de Programación Orientada a Objetos

<https://youtu.be/5flyAn9IvAU>

Link a lectura complementaria

Principios fundamentales de la Programación Orientada a Objetos

<https://desarrolloweb.com/manuales/programacion-orientada-objetos-dotnet.html>

Temario

Unidad 5

a) Pruebas Unitarias en Java

b) Introducción a JUnit

c) Utilización de Fixtures en las unidades de prueba

Unidad 5

Pruebas Unitarias en Java

Prueban una funcionalidad única y se basan en el principio de responsabilidad única (la S de los principios de diseño S.O.L.I.D.)

Las pruebas unitarias son una forma de probar el buen funcionamiento de un modulo o una parte de sistema con el fin de asegurar el correcto funcionamiento de todos los modelos por separado y evitar si errores futuros en el momento de la integración de todas sus partes.

Unidad 5

Los tests unitarios prueban las funcionalidades implementadas en el SUT (System Under Test). Para los desarrolladores Java, el SUT será la clase Java.

Los tests unitarios deben cumplir las siguientes características:

Principio F.I.R.S.T.

Fast: Rápida ejecución.

Isolated: Independiente de otros test.

Repeatable: Se puede repetir en el tiempo.

Self-Validating: Cada test debe poder validar si es correcto o no a sí mismo.

Timely: ¿Cuándo se deben desarrollar los test? ¿Antes o después de que esté todo

Unidad 5

Ventajas de JUnit

- Es gratuito, open source (código abierto) y puede ser utilizado en desarrollos comerciales.
- Se encuentra actualmente bien mantenido y en constante mejora.
- Reglas de diseño de JUnit pueden crear una biblioteca modularizada de casos de pruebas para cada clase bajo construcción
- Permite un desarrollo de código mas rápido y de mayor calidad.
- Comprueba resultados operativos propios y para proveer una retroalimentación puntual.
- Las pruebas son escritas utilizando Java

Unidad 5

Limitaciones o Desventajas

- Reglas de diseño JUnit tienden a dirigir a un numero masivo de pruebas de clases
- El código de las pruebas no se aislara de los datos de pruebas
- JUnit no hace pruebas unitarias en interfaces Swing, JSPs, HTML.
- Las pruebas de JUnit no pueden reemplazar las pruebas funcionales.
- Las pruebas unitarias no pueden ser código Java del lado del servidor.

Unidad 5

Introducción a JUnit

JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, **para** poder evaluar si el funcionamiento de cada uno de los métodos de la clase se comporta como se espera.



Se basa en anotaciones:

- **@Test**: indica que el método que la contiene es un test: expected y timeout.
- **@Before**: ejecuta el método que la contiene justo antes de cada test.
- **@After**: ejecuta el método que la contiene justo después de cada test.
- **@BeforeClass**: ejecuta el método (estático) que la contiene justo antes del **primer test**.
- **@AfterClass**: ejecuta el método (estático) que la contiene justo después del **último test**.
- **@Ignore**: evita la ejecución del tests. No es muy recomendable su uso porque puede ocultar test fallidos. Si dudamos si el test debe estar o no, quizás borrarlo es la mejor de las decisiones.

Unidad 5

Las condiciones de aceptación del test se implementa con los asserts. Los más comunes son los siguientes:

- **assertTrue/assertFalse** (condición a testear): Comprueba que la condición es cierta o falsa.
- **assertEquals/assertNotEquals** (valor esperado, valor obtenido). Es importante el orden de los valores esperado y obtenido.
- **assertNull/assertNotNull (object)**: Comprueba que el objeto obtenido es nulo o no.
- **assertSame / assertNotSame(object1, object2)**: Comprueba si dos objetos son iguales o no.
- **fail()**: Fuerza que el test termine con fallo. Se puede indicar un mensaje.

Unidad 5

Utilización de Fixtures en las unidades de prueba

Cuando hablamos de unit tests debemos hablar de “Mocking”, este es una de las habilidades principales que debemos tener a la hora de hacer tests en nuestras aplicaciones.

Al momento de desarrollar aplicaciones las dividimos en capas, web, negocio y datos, entonces al escribir tests unitarios en una capa, no queremos preocuparnos por otra, así que la simulamos, a este proceso se le conoce como Mocking.

Unidad 5

Mocking

Un objeto mock es un proveedor de datos que cuando se ejecuta la aplicación el componente se conectará a una base de datos y proveerá datos reales, pero cuando se ejecuta un test unitario lo que buscamos es aislarlo y para esto necesitamos un objeto mock que simulará la fuente de datos, esto asegurará que las condiciones de prueba sean siempre las mismas.

Material complementario de la unidad

Link a video relacionado

Test en Java con JUnit para comprobación de passwords

<https://youtu.be/kt1jPjAr34Y>

Link a lectura complementaria

Java Mockito y los Mock Object

<https://www.arquitecturajava.com/java-mockito-y-los-mock-objects/>