

Guía de configuración y ejecución del corrector automático para el proyecto de POO

Fecha: 14/3/2024

Versión: 1.0

Autor: Juan Carlos Cruellas

1.	QUÉ DEBÉIS LEER DE ESTE DOCUMENTO	3
2.	INTRODUCCIÓN AL PROCESO DE CONFIGURACIÓN: QUÉ ES NECESARIO TENER	3
3.	TAREAS DE CONFIGURACIÓN PREVIAS AL TRABAJO EN EL PROYECTO	4
	APÉNDICE: ALGUNAS PINCELADAS DE APACHE MAVEN	13

1. Qué debéis leer de este documento

Es imprescindible que leáis las secciones 2, 3 y 4.

Si no sabéis qué es Maven y para qué sirve, debéis leer también el Apéndice de este documento.

Las secciones 2 y 3 tienen que ver con la configuración de vuestro entorno de trabajo para que el corrector automático pueda ejecutarse sin problemas.

Es muy importante que leáis la sección 4, que muestra:

1. En qué orden debéis desarrollar vuestro proyecto
2. Cómo debéis ejecutar el corrector automático
3. Qué debéis hacer en caso de que el corrector automático notifique errores en vuestro código

2. Introducción al proceso de configuración: qué es necesario tener

Al publicar el enunciado de una parte del proyecto (como se os comentó el primer día, el proyecto tiene dos) se os hace entrega de un proyecto NetBeans que incorpora:

1. Una serie de clases, distribuidas en packages (en la carpeta “Source Packages”), **cuyo código deberéis completar** para que cumpla con lo especificado en el javadoc que acompaña al enunciado.
2. Una serie de clases hechas (en la carpeta “Test Packages”) **que constituyen el corrector automático**. Estas clases os permitirán, como permiten los correctores automáticos de la colección de problemas que encontraréis en el metacurso de la asignatura, **corregir en tiempo real los métodos que vayáis desarrollando y os mostrarán la nota que se os otorgará en la parte de evaluación objetiva del proyecto**. De esta forma, en el momento de la entrega conoceréis dicha calificación.

Quienes tengáis ordenador en casa y queráis desarrollar en él el proyecto de la asignatura con el corrector automático, deberéis asegurarnos de tener en sus máquinas los siguientes componentes:

- **El Java Development Kit (JDK) de Java**. La versión instalada debe ser la 1.8 o superior (la última existente cuando se escribió este documento es la 17). La sintaxis que se presentará en la asignatura será un subconjunto de la versión 1.8 y la compatibilidad hacia atrás de Java asegura que cualquier programa desarrollado con sintaxis de la versión 1.8 se ejecuta sin problemas en cualquier versión posterior.
- **El Entorno de Desarrollo Integrado (Integrated Development Environment -IDE- en inglés) NetBeans**. La última versión estable de este programa, en el momento en que se escribió este documento es NetBeans 12.6.

NOTAS MUY IMPORTANTES:

NOTA 1: Antes de proceder a instalar NetBeans y a ejecutarlo por primera vez, leed el siguiente documento

<https://blogs.apache.org/netbeans/entry/what-s-nb-javac-in>

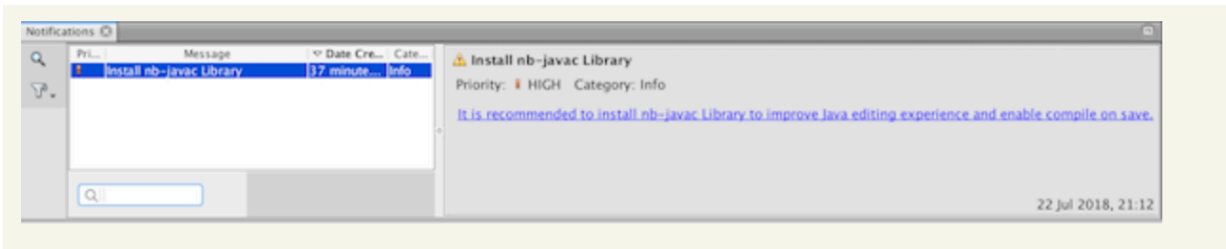
NOTA 2: Después de instalar NetBeans, debe tenerse mucho cuidado al ejecutarlo por primera vez. Como habéis leído en el documento anteriormente

mencionado, NetBeans pregunta si se desea instalar el plugin nb-javac. **ES MUY IMPORTANTE CONTESTAR QUE SÍ.**

NOTA 3: Si se ha instalado ya NetBeans, ES MUY IMPORTANTE COMPROBAR QUE SE HA INSTALADO dicho plugin. Para ello id a

Window -> IDE Tools -> Notifications

Si NetBeans os muestra en la parte inferior del IDE lo siguiente, proceded a instalarlo siguiendo el enlace de la derecha, siguiendo las instrucciones que os irá dando el mismo IDE.



De no tener instalado el mencionado plugin, podréis tener problemas de diferentes tipos (no detección de las clases de test, etc).

Finalmente, debéis saber que para que **el proyecto que se os entrega es un proyecto Maven**, y para que **el corrector automático funcione es necesario realizar unas pequeñas tareas de configuración.**

3. Tareas de configuración previas al trabajo en el proyecto

Junto con el proyecto NetBeans, el enunciado y el javadoc con la especificación de todas las clases que tendréis que desarrollar, se os entrega el siguiente archivo JAR:

BaseCorrectorDAC-2022_2023on.jar

Este JAR contiene una serie de clases que el corrector automático necesita para trabajar correctamente.

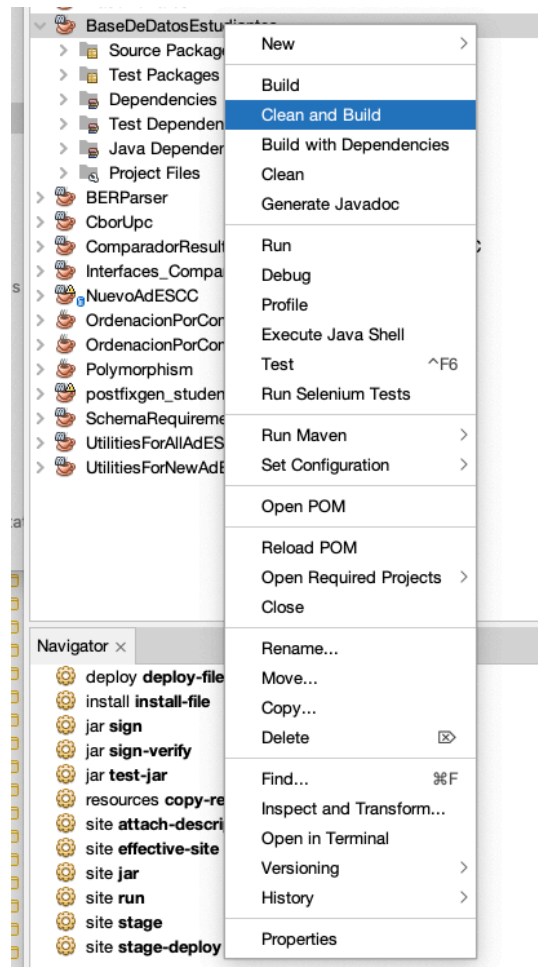
Este JAR NO ha sido publicado en el repositorio central de Maven.

NOTA IMPORTANTE: Si no sabéis qué es Maven, leed el **“Apéndice: Algunas pinceladas de Apache Maven”** al final de este documento antes de seguir.

Para dejar el proyecto NetBeans que se os entrega preparado para que empecéis a desarrollar vuestro proyecto añadiendo código a las clases incompletas, debéis realizar los siguientes pasos:

1. Descomprimid el archivo zip que contiene a dicho proyecto NetBeans.
2. Abrid el proyecto con NetBeans.
3. Haced un “Clean and Build” del proyecto. Eso borrará todos los archivos .class con bytecode resultantes de compilaciones anteriores e intentará regenerarlos todos y construir el proyecto. Esta operación se realiza seleccionando el proyecto en el navegador de proyectos de NetBeans, pulsando el botón derecho del ratón y seleccionando “Clean and Build”, tal y como se muestra en la siguiente figura:

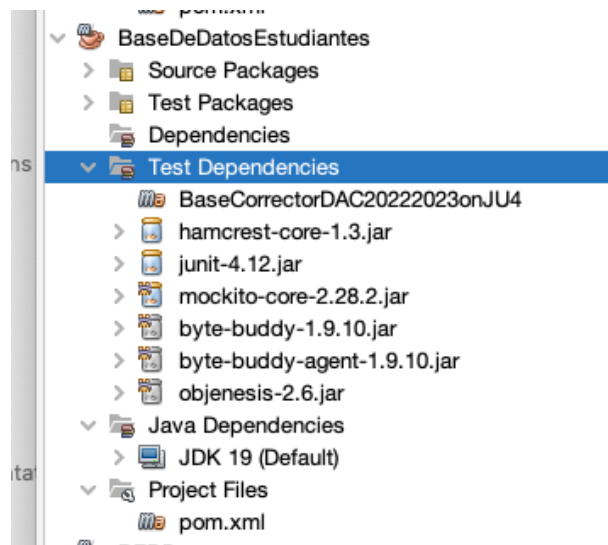
NOTA IMPORTANTE: las figuras mostradas en este documento corresponden a un proyecto de un curso anterior, por lo que algunos nombres que veréis en ellas (Decathlon1ParaP..., por ejemplo) NO corresponden con lo que veréis en vuestro proyecto. Tened esto en cuenta cuando analicéis las figuras que siguen.



4. El proceso acabará notificando un error. Ese error se produce porque, como se ha comentado, el archivo `BaseCorrectorDAC-2022_2023on.jar` **no ha sido publicado en el repositorio central de Maven y tampoco lo tenéis en vuestro repositorio local**. Para que todo funcione correctamente debéis publicarlo en vuestro repositorio local. Observad que una de las líneas que os devuelve NetBeans por la consola es algo como lo que sigue:

```
Failed to execute goal on project BaseDeDatosEstudiantes: Could not
resolve dependencies for project
edu.upc.etsetb.poo:BaseDeDatosEstudiantes:jar:1.0-SNAPSHOT: Could not
find artifact edu.upc.ac:BaseCorrectorDAC:jar:2022_2023on in central
(https://repo.maven.apache.org/maven2) -> [Help 1]
```

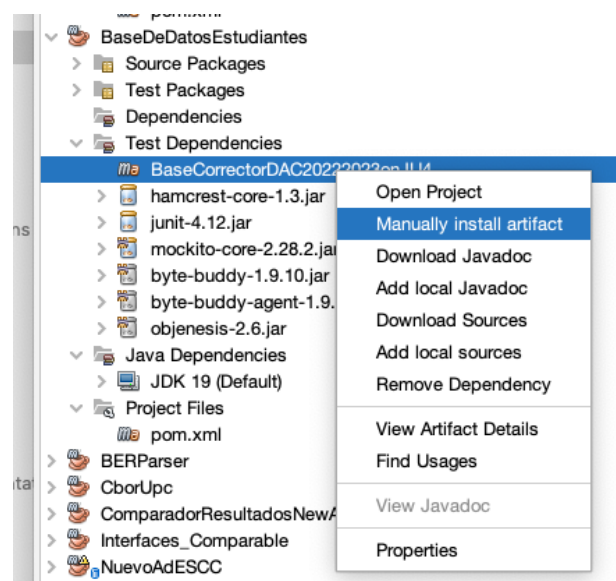
Observad asimismo que, si desplezáis la carpeta “Test Dependencies” del navegador de proyectos, veréis algo similar a lo que muestra la siguiente figura:



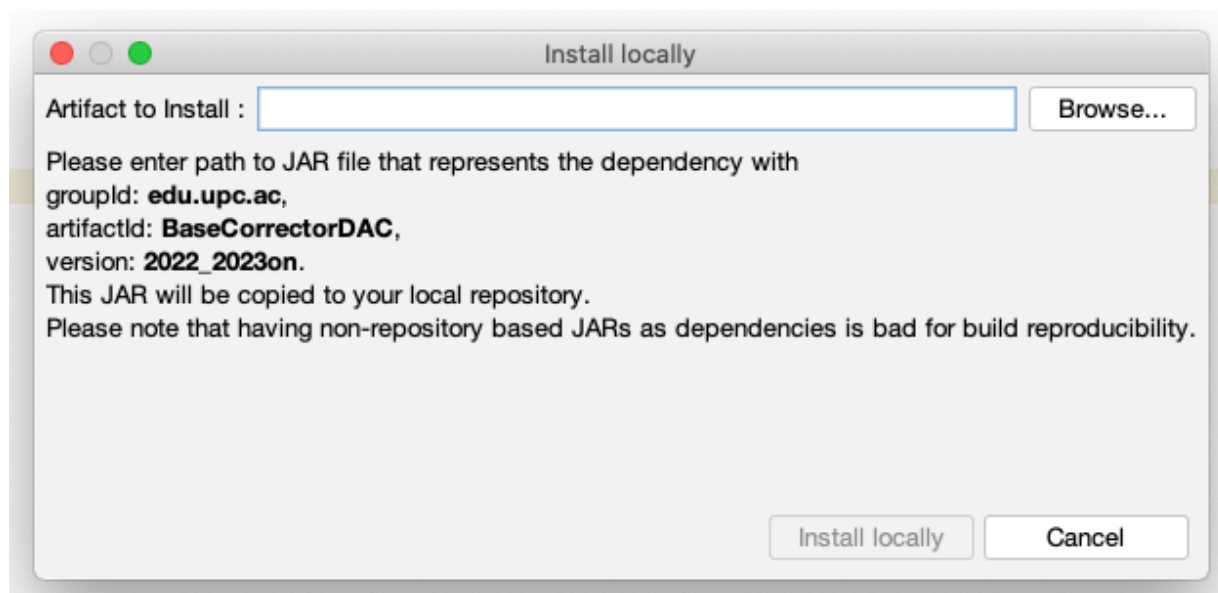
Observad que a la izquierda de `BaseCorrectorDAC20222023onJU4`, no se muestra ningún `>` (el símbolo que indica que se despliega una lista con información). Eso indica que NetBeans no sabe dónde encontrar dicho módulo. En otras palabras, esta figura muestra que los tests (las pruebas que realiza el corrector automático) dependen del archivo `BaseCorrectorDAC-2022_2023on.jar` que, tal y como ha notificado NetBeans, no ha sido encontrado ni en el repositorio maven central ni en el local.

Para solventar este problema deberéis depositar dicho archivo en el repositorio local

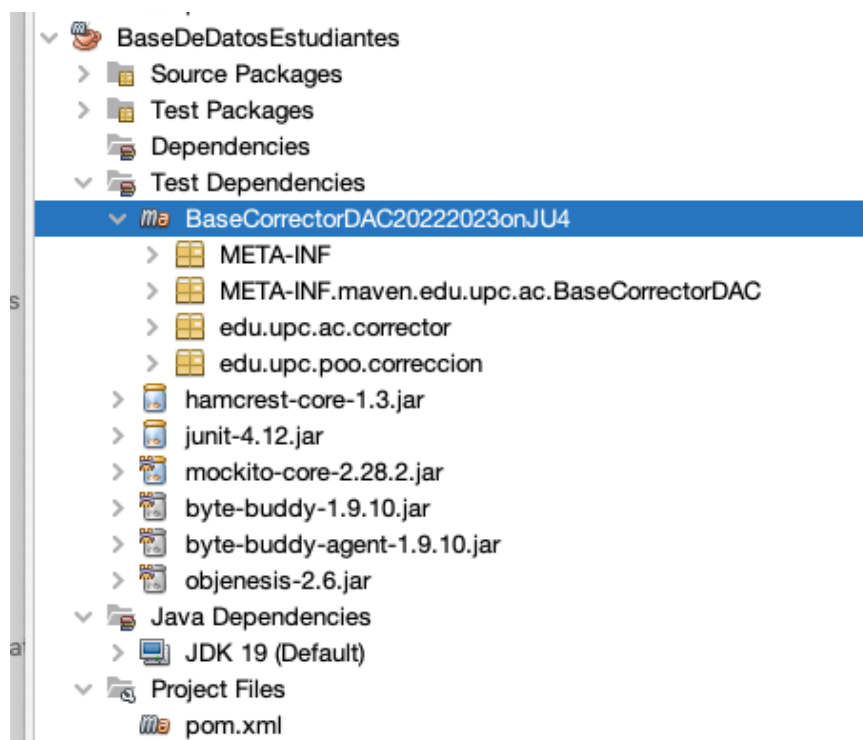
5. Para depositar el JAR en el repositorio local, y que así Maven pueda encontrarlo, seleccionad **BaseCorrectorDAC-2022_2023on** en la carpeta “Test Dependencies” del navegador de proyecto de NetBeans. Pulsad el botón derecho del ratón y seleccionad “Manually install artifact”, tal y como se muestra en la siguiente figura:



6. NetBeans os muestra entonces un cuadro de diálogo para que le paséis el archivo que queréis depositar en el repositorio local, tal y como muestra la figura que sigue.

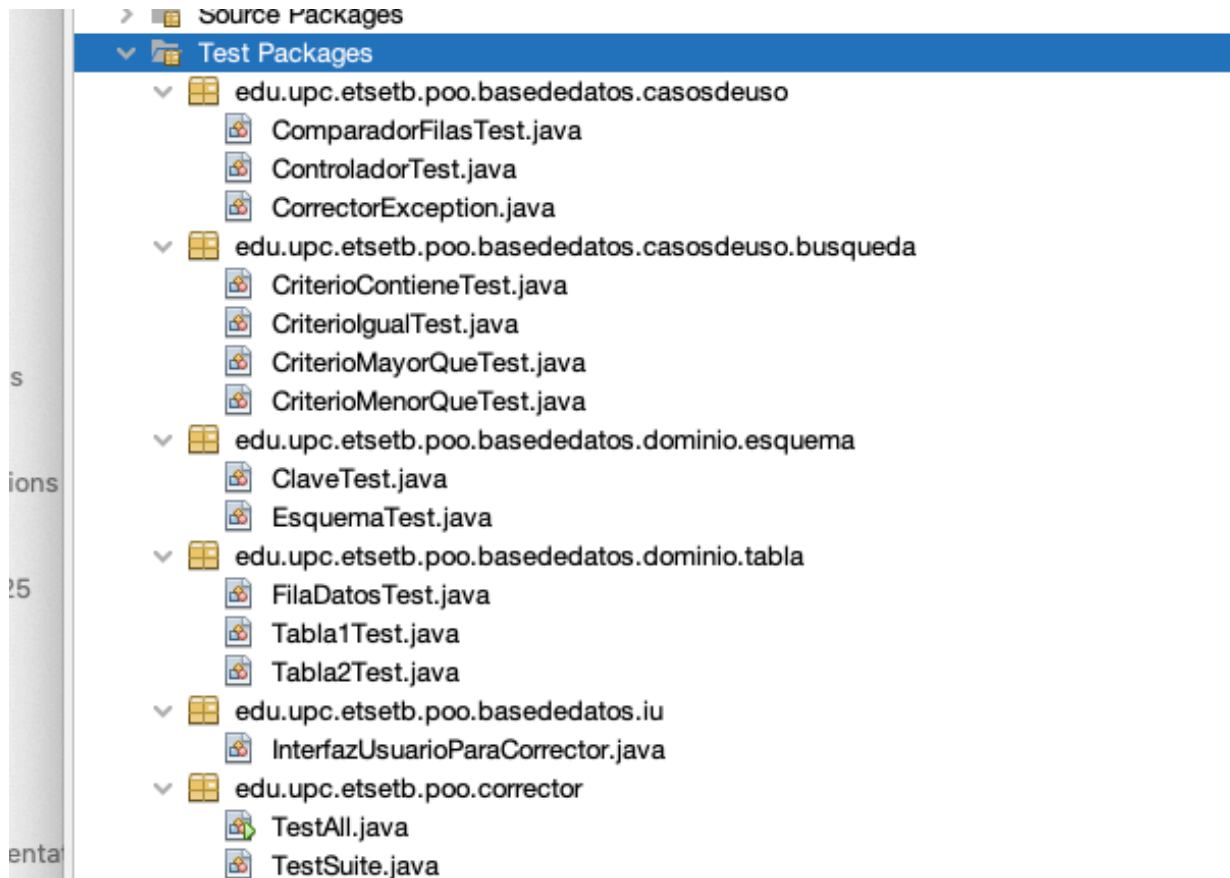


7. Pulsad el botón “Browse...” y se os abrirá el navegador de directorios de vuestra máquina. Id al directorio donde esté el archivo **BaseCorrectorDAC-2022_2023on.jar**, seleccionadlo y pulsad el botón “Install locally”.
8. Cuando lo hagáis, Maven procederá a depositar el mencionado JAR en vuestro repositorio local y podrá acabar de construir el proyecto (con vuestras clases incompletas) sin problemas. Ahora, si desplegáis la carpeta “Test Dependencies” observaréis que a la izquierda de **BaseCorrectorDAC20222023onJU4** ha aparecido un ➤. Si lo pulsáis, se os desplegará una lista de ítems como la que se muestra en la figura:



9. La aparición de los packages META-INF a edu.upc.poo.correccion indica que el package necesario para el corrector ha sido instalado correctamente en el proyecto Netbeans y en vuestro repositorio maven local. **Desarrollando y corrigiendo el proyecto con el corrector automático**

Si desplegáis los packages que hay en la carpeta “Test Packages” veréis todas las clases (**ya hechas y que no debéis alterar**) del corrector automático, tal y como se muestra a continuación:



NOTA IMPORTANTE: observad nuevamente que veréis TestAll y TestSuite en vuestro NetBeans, puesto que son las clases que orquestan el trabajo del resto de clases del corrector. Veréis otras clases (ClaveTest, EsquemaTest, etc): cada una de esas clases corregirá a la clase correspondiente que tenéis que desarrollar (ClaveTest corregirá vuestra clase Clave, EsquemaTest corregirá vuestra clase Esquema y así sucesivamente).

En definitiva, por cada clase que hay en los packages de la carpeta “Source Packages” hay una clase correctora en el mismo package de la carpeta “Test Packages”, cuyo nombre es el mismo que el de la clase que tenéis que completar acabado en “Test”, excepto para las clases XXXException y las clases cuyo código completo se os da.

Cada una de estas clases corrige el código de la correspondiente clase de la carpeta “Source Packages”.

Además, en el package **edu.upc.etsetb.poo.corrector** de la carpeta “Test Packages” tenéis dos clases adicionales:

- TestSuite. Este archivo enumera a las clases test antes mencionadas. Esta enumeración determina el orden en que se ejecutarán sus códigos, esto es, el orden en el que vuestras clases se corregirán. **Determina por tanto el orden en el que deberéis desarrollar vuestras clases.** Es el mismo orden que se os sugiere en el enunciado del problema. **Es, por tanto, muy importante que desarrolléis el código siguiendo dicho orden.**
- TestAll. Este archivo contiene el main() del corrector automático. **Es el programa que hay que ejecutar -tal y como ejecutáis cualquier clase Java que tenga el método main()- para que el corrector automático corrija vuestro código y os dé los detalles de la corrección:** para cada clase

corregirá cada método; para cada uno de ellos os dirá qué pruebas dicho método ha superado con éxito, qué pruebas no, y os dará la nota obtenida. Al final os hará un resumen de las notas obtenidas en cada clase y la nota total.

NOTA MUY IMPORTANTE:

Para avanzar sin problemas en el desarrollo **es extremadamente importante que cada vez que finalicéis el desarrollo de UN MÉTODO lancéis la ejecución de TestAll.**

Así, si tenéis algún error en él, el corrector os lo notificará y con los mensajes que os muestre, podréis tener una idea de las posibles razones del error.

Cuando eso ocurra, si una relectura del código no os permite identificar el problema, debéis poner un breakpoint al comienzo del método que acabáis de desarrollar, **y lanzar una sesión de depuración con TestAll (Debug File)**. El corrector se detendrá en esa línea y a partir de ahí podréis analizar la evolución de vuestro código al ser ejecutado para identificar el error (o los errores) que tengáis. Cuando creáis que los habéis resuelto, lanzad una ejecución del corrector. Repetid esta pauta hasta que el corrector os diga que todo en el método es correcto. Solo entonces deberíais pasar a desarrollar un nuevo método.

A continuación, se dan algunos detalles sobre los resultados presentados por el corrector.

FASE DE INICIO:

1. Cuando abráis el proyecto NetBeans y completéis su configuración tal y como se comenta en la sección 3 de este documento, haced que NetBeans lleve a cabo la operación "Clean and Build". Eso borra cualquier archivo .class que el proyecto haya generado antes y procederá a recompilar todos los archivos de código fuente y a reconstruir todo el proyecto.
2. A continuación, ejecutad el archivo TestAll (Run File). El corrector comenzará a mostraros líneas como las que siguen:

```
Running edu.upc.etsetb.poo.corrector.TestSuite
```

```
Corrigiendo clase Clave
```

```
Clave::Clave(nombre). Valor: 3.0

*** Se ha capturado una excepción que probablemente ha sido lanzada por tu
código. Mira la traza para detectar en qué punto ha sido creada y lanzada...

    UnsupportedOperationException: Clave::Clave(nombrea) no implementado.
    at edu.upc.etsetb.poo.basededatos.dominio.esquema.Clave.<init>(Clave.java:38)
    at
    edu.upc.etsetb.poo.basededatos.dominio.esquema.ClaveTest.test01_ConstructorCon
    1Argumento(ClaveTest.java:85)
    at edu.upc.etsetb.poo.corrector.TestAll.main(TestAll.java:60)
```

```
Puntos obtenidos: 0.0. Puntos acumulados: 0.0
```

El mensaje inicial indica que alguien ha lanzado una excepción, previsiblemente vuestro código.

El texto en negrita informa de la excepción lanzada (UnsupportedOperationException) y un mensaje que explica la razón por la que se lanzó (en este caso que el método constructor de Clave NO lo habéis implementado todavía: todos los método que debéis implementar contienen,

de entrada una instrucción que lanza esta excepción para que no olvidéis implementarlos).

Las siguientes líneas comienzan por “at”. Cada una de esas líneas dan detalles de la secuencia de invocaciones que han generado este lanzamiento.

Tened esto en cuenta cada vez que os aparezcan esos listados. ¡¡¡¡Si la primera línea que comienza con “at” apunta a una de vuestras clases, es que tenéis un problema en vuestro código!!!!.

En el caso mostrado, estos mensajes dicen:

- Que el corrector ha corregido el código del método Clave(...) de Clave.
- Que este método ha lanzado una excepción en la línea 38 del archivo Clave.java.
- Que este método se ha invocado desde la línea 85 de la clase ClaveTest.
- Que este método se ha invocado desde la línea 60 de la clase TestAll.
- Que el motivo por el que ha lanzado la excepción es que el método no se ha implementado todavía
- Que en consecuencia no ha otorgado ningún punto.

Observad ahora el código del constructor en Clave.java:

```
public Clave(String nombre) {  
    throw new UnsupportedOperationException("Clave::Clave(nombre) no  
    implementado.");  
}
```

La única instrucción que hay es la que hace que el corrector presente el mensaje anterior.

En vuestro proyecto, todos los métodos que tenéis que desarrollar tienen instrucciones similares.

Vuestro trabajo consistirá en sustituir estas instrucciones por código que se implemente lo especificado en la documentación (javadoc).

FASE DE TRABAJO.

Una vez finalizada la fase de configuración ya podéis empezar a desarrollar el código de vuestro proyecto (fase de trabajo).

Durante esta fase, podéis trabajar en 2 modos:

- Ejecutando la clase TestAll que encontraréis en el package de **Tests** edu.upc.etsetb.poo.corrector, utilizando “Run File” o “Debug File” (según queráis ejecutar o depurar (poniendo, por ejemplo, un breakpoint en el método que estáis generando)).
- Ejecutando de forma individual cada una de las clases de test (ClaveTest, EsquemaTest) utilizando “Test File” o “Debug Test File”.

MODO 1: EJECUTANDO TestAll

Ahora comenzad sustituyendo el código del Clave(String nombre) anterior por vuestro código.

Cuando hayáis acabado de hacerlo, lanzad una nueva ejecución de **TestAll**. Imaginad ahora que habéis cometido algún error y que el comportamiento del

método no es el especificado en el javadoc. En tal caso el corrector podría generar una salida como la que se muestra a continuación:

```
Corrigiendo clase Clave
```

```
Clave::Clave(nombre). Valor: 3.0
Test 1: comprobación de existencia de atributo de nombre 'nombre'
(0.75)
El atributo nombre tiene un valor null. Debería
tener el valor: claveUnica;
Test 3: comprobación de existencia de atributo de nombre 'unica' (0.75)
Test 4: comprobación del valor asignado por el corrector al atributo de
nombre 'unica' (0.75)
Puntos obtenidos: 2.25. Puntos acumulados: 2.25
```

Para cada método el corrector realizará una serie de pruebas.

Si el código no se ajusta a la especificación, debajo de una indicación de Test aparece una línea indentada con un mensaje que indica la razón por la que el corrector considera que la prueba ha fallado. Cuando eso sucede el corrector no acumula puntos. Con esta información debéis averiguar qué parte de vuestro código está fallando exactamente, el por qué del error y debéis modificarlo hasta que supere la prueba. Al final el corrector presenta el número de puntos obtenidos por la corrección del método y el número de puntos acumulados en la corrección de la clase en cuestión.

A continuación se muestra el comportamiento del corrector cuando el método ha sido bien programado:

```
Resumen de notas obtenidas en corrección automática:
```

```
Nota en clase Controlador: 0.0 (Porcentaje en nota final: 27.5%). Contribución
a nota final: 0.0
Nota en clase CriterioMayorQue: 0.0 (Porcentaje en nota final: 7.5%).
Contribución a nota final: 0.0
Nota en clase Tabla2: 0.0 (Porcentaje en nota final: 7.5%). Contribución a
nota final: 0.0
Nota en clase Clave: 2.25 (Porcentaje en nota final: 5.0%). Contribución a
nota final: 0.1125
Nota en clase ComparadorFilas: 0.0 (Porcentaje en nota final: 7.5%).
Contribución a nota final: 0.0
Nota en clase FilaDatos: 0.0 (Porcentaje en nota final: 7.5%). Contribución a
nota final: 0.0
Nota en clase Tabla1: 0.0 (Porcentaje en nota final: 7.5%). Contribución a
nota final: 0.0
Nota en clase CriterioIgual: 0.0 (Porcentaje en nota final: 7.5%).
Contribución a nota final: 0.0
Nota en clase Esquema: 0.0 (Porcentaje en nota final: 7.5%). Contribución a
nota final: 0.0
Nota en clase CriterioMenorQue: 0.0 (Porcentaje en nota final: 7.5%).
Contribución a nota final: 0.0
Nota en clase CriterioContiene: 0.0 (Porcentaje en nota final: 7.5%).
Contribución a nota final: 0.0
```

```
Nota final de corrección automática: 0.113
```

```
Listado de errores detectados:
```

```
Errores detectados por clase ClaveTest
El atributo nombre tiene un valor null. Debería
tener el valor: claveUnica;
```

Continuad modificando vuestro código y ejecutando el corrector hasta que el corrector indique que no hay errores (o hasta que llegue el momento de realizar la entrega del proyecto). Con el corrector sabréis en todo momento la nota objetiva que vuestro proyecto tendrá.

MODO 2: EJECUTANDO LAS CLASES DE TEST INDIVIDUALES

En ese modo, el corrector solo ejecuta las pruebas correspondientes a UNA clase (normalmente la clase de test correspondiente a la clase que estéis construyendo).

Abrid en el editor la clase `ClaveTest`. Ahora pulsad el botón derecho del ratón; os aparecerá un menú contextual. En él encontraréis las opciones “Test File” y “Debug Test File”. La primera lanza la ejecución de las pruebas programadas en `ClaveTest`. La segunda lanza una sesión de depuración. En este caso la ejecución se detendrá allí donde hayáis puesto un breakpoint (típicamente, o bien en alguna línea del método de `Clave` que estéis depurando, o en alguna línea del método de `ClaveTest` que corrige dicho método).

El corrector va acumulando puntos con las pruebas superadas satisfactoriamente, y al final de la corrección de la clase, muestra la puntuación obtenida.

Cuando todas las pruebas se han superado satisfactoriamente, la calificación otorgada a la clase es 10.0.

Apéndice: Algunas pinceladas de Apache Maven

Apache Maven es una herramienta de código libre desarrollada en el seno de la fundación Apache creado inicialmente para dar soporte a las tareas de compilación y construcción de programas Java. No deberíais pensar que llevar a cabo estas tareas manualmente es fácil. Pensad que las aplicaciones Java del mundo real pueden tener miles y miles de líneas de código y cientos y cientos de clases distribuidas en multitud de módulos compilados y comprimidos (los archivos JAR -Java Archive), creados por grupos de desarrolladores distintos, que hay que integrar en único programa.

Antes del desarrollo de Maven y de herramientas similares, para construir un programa Java:

1. Era preciso descargarse uno a uno esos archivos JAR generados por diversos grupos, lo que conllevaba visitar multitud de páginas web donde dichos módulos se ponían a disposición de los desarrolladores;
2. Había que decidir dónde dejarlos en la máquina en la que se realizaba el desarrollo y
3. Había que configurar el IDE utilizado para que trabajase, cuando le hiciera falta, con esos archivos JAR descargados.

Podéis imaginar que si hablamos de decenas y decenas de archivos JAR, el trabajo resultaba tedioso. Y eso sin hablar de las versiones: es muy frecuente que esos módulos desarrollados por terceros evolucionen y sus desarrolladores generen nuevas versiones (pensad, por ejemplo en una librería criptográfica como bouncycastle -<https://www.bouncycastle.org/>-; es una librería en constante evolución porque aparecen nuevos algoritmos criptográficos constantemente). Cuando aparece el problema del versionado, la construcción de programas reales se hace todavía más compleja.

Maven facilita enormemente este proceso. La infraestructura Maven está formada por varios elementos fundamentales:

1. Un REPOSITORIO CENTRAL PÚBLICO de libre acceso, que puede visitarse en <https://repo.maven.apache.org/maven2/>. En ese repositorio los desarrolladores de herramientas de código libre depositan sus librerías en módulos JAR. En el repositorio el código se organiza en un árbol de directorios resultante de mapear NOMBRES DE PACKAGES (navegad y visitad, por ejemplo, el directorio `org` y dentro de él, entrad en el subdirectorio `bouncycastle`: eso os dará una idea más precisa de lo que podéis encontrar allí).
2. Herramientas de búsqueda en ese repositorio central (<https://mvnrepository.com/repos/central>, <https://search.maven.org/>) que facilitan la búsqueda de JARs específicos.
3. El programa Maven propiamente dicho, que quienes desarrollan programas Java tienen en sus máquinas.
4. Un repositorio LOCAL que Maven crea en las máquinas en que se instala. En general suele ser un árbol de directorios cuya raíz es `~/.m2`. El `.` Del principio lo identifica como un directorio oculto. Buscad en google cómo ver los archivos y directorios ocultos en la máquina con la que trabajáis. El lugar donde Maven crea por defecto dicho directorio varía de máquina en máquina. Buscad los detalles en google para la máquina con la que trabajéis (Linux, Windows o Mac).
5. Para cada programa Java cuya construcción gestiona Maven, se crea un archivo XML que, entre otras cosas, enumera los archivos JAR que participan en la

construcción del programa final: son archivos de los que dicho programa DEPENDE. En ese archivo se denomina "Project Object Model" o POM. Los IDE como NetBeans permiten generar de forma automática una buena parte de dichos archivos, al menos la fundamental.

Cuando tiene que construirse un programa, Maven analiza el archivo POM, y entre otras cosas, pasa revista a las DEPENDENCIAS (esto es, la lista de archivos JAR que el programa necesita).

Cuando identifica una dependencia en el POM, Maven lo busca en el REPOSITORIO LOCAL.

Si lo encuentra, lo integra en la construcción del programa.

Si NO lo encuentra en el repositorio local, Maven se conecta automáticamente al REPOSITORIO CENTRAL y, si lo encuentra, lo descarga, LO DEPOSITA EN EL REPOSITORIO LOCAL para evitar nuevas conexiones innecesarias en el futuro, y lo integra en la construcción del programa.

Obviamente, si tampoco lo encuentra en el repositorio central, Maven da un mensaje de error y el proceso de construcción finaliza con un error.

Para generar la parte del archivo POM que lista las dependencias del proyecto, los desarrolladores hacen uso de los buscadores en el repositorio central Maven como se explica a continuación.

Imaginad que tenéis que desarrollar un programa de gestión de firma electrónica, por ejemplo. Dicho programa necesitará disponer de módulos que implementen los algoritmos criptográficos más comunes. Como se ha dicho antes, una de las más populares es BouncyCastle. Para incorporar la dependencia en nuestro POM, buscaríamos bouncycastle usando el primer buscador mencionado en el punto 2. Dicho buscador nos devuelve los resultados que se muestran en la figura siguiente:

Repository

Central 143

Sonatype 30

Spring Lib M 29

Spring Plugins 17

Redhat GA 14

ICM 7

IBiblio 6

JCenter 6

Group

org.bouncycastle 77

bouncycastle 15

org.apache 8

io.github 6

com.liferay 4

com.sshools 4

janstey.sources 4

maven-repository.org 4

Category

Encryption Lib 5

Maven Plugins 1

License

BouncyCastle 75

Apache 47

LGPL 7

EPL 6

MIT 6


BSD 5

EDL 5

GPL 4

Found 192 results

Sort: **relevance** | popular | newest



1. Bouncy Castle Provider


org.bouncycastle » bcprov-jdk16

552 usages

BouncyCastle

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.6.

Last Release on Feb 23, 2011



2. Bouncy Castle Provider


org.bouncycastle » bcprov-jdk15on

2,875 usages

BouncyCastle

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 and up.

Last Release on Dec 1, 2021



3. Legion of The Bouncy Castle Java Cryptography APIs


bouncycastle » bcprov-jdk15

197 usages

BouncyCastle

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. The package is organised so that it contains a light-weight API suitable for use in any environment (including the newly released J2ME) with the additional infrastructure to conform the algorithms to the JCE framework.

Last Release on Sep 26, 2008



4. Bouncy Castle PKIX, CMS, EAC, TSP, PKCS, OCSP, CMP, and CRMF APIs


org.bouncycastle » bcpkix-jdk15on

1,913 usages

BouncyCastle

The Bouncy Castle Java APIs for CMS, PKCS, EAC, TSP, CMP, CRMF, OCSP, and certificate generation. This jar contains APIs for JDK 1.5 and up. The APIs can be used in conjunction with a JCE/JCA provider such as the one provided with the Bouncy Castle Cryptography APIs.

Last Release on Dec 1, 2021



5. Legion of The Bouncy Castle Java Cryptography APIs


bouncycastle » bcprov-jdk14

90 usages

BouncyCastle

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. The package is organised so that it contains a light-weight API suitable for use in any environment (including the newly released J2ME) with the additional infrastructure to conform the algorithms to the JCE framework.

Last Release on Oct 1, 2008



6. JClouds BouncyCastle EncryptionService Module

org.apache.jclouds.driver » jclouds-bouncycastle


20 usages

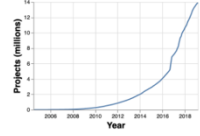
Apache

jclouds bouncycastle EncryptionService Module

Last Release on Sep 10, 2021

De entrada, daos cuenta de la cantidad de resultados (y la figura no muestra todos los devueltos, de hecho ¡hay 10 páginas de resultados!). Si ahora seleccionamos el primero de ellos (Bouncy Castle Provider, bcprov-jdk1.6), la herramienta presenta el siguiente resultado:



Indexed Artifacts (25.7M)



Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities
- Dependency Injection
- Embedded SQL Databases
- HTML Parsers
- HTTP Clients
- I/O Utilities
- JDBC Extensions
- JDBC Pools
- JPA Implementations
- JSON Libraries
- JVM Languages
- Logging Frameworks
- Logging Bridges
- Mail Clients
- Maven Plugins
- Mocking
- Object/Relational Mapping
- PDF Libraries

Top Categories

<https://mvnrepository.com>

Home » [org.bouncycastle](#) » [bcprov-jdk16](#)



Bouncy Castle Provider

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.6.

License	BouncyCastle
Categories	Encryption Libraries
Tags	encryption
Used By	552 artifacts

Central (6)
Redhat GA (1)
Redhat EA (1)
ICM (2)
EBIPublic (3)

Version	Vulnerabilities	Repository	Usages	Date
1.46	2 vulnerabilities	Central	452	Feb, 2011
1.45	2 vulnerabilities	Central	89	Jan, 2010
1.44	2 vulnerabilities	Central	15	Oct, 2009
1.43	2 vulnerabilities	Central	10	Jun, 2009
1.40	2 vulnerabilities	Central	2	Jul, 2009
1.38	2 vulnerabilities	Central	2	Jul, 2009

El repositorio nos muestra hasta 6 versiones diferentes de esta herramienta. Seleccionando la primera (1.46), el buscador nos devuelve:

Indexed Artifacts (25.7M)

Popular Categories

- Aspect Oriented
- Actor Frameworks
- Application Metrics
- Build Tools
- Bytecode Libraries
- Command Line Parsers
- Cache Implementations
- Cloud Computing
- Code Analyzers
- Collections
- Configuration Libraries
- Core Utilities
- Date and Time Utilities
- Dependency Injection
- Embedded SQL Databases
- HTML Parsers
- HTTP Clients
- I/O Utilities
- JDBC Extensions
- JDBC Pools
- JPA Implementations
- JSON Libraries
- JVM Languages
- Logging Frameworks
- Logging Bridges
- Mail Clients
- Maven Plugins
- Mocking
- Object/Relational Mapping
- PDF Libraries

Top Categories

[Home](#) » [org.bouncycastle](#) » [bcprov-jdk16](#) » [1.46](#)

Bouncy Castle Provider » 1.46

The Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.6.

License	BouncyCastle
Categories	Encryption Libraries
HomePage	http://www.bouncycastle.org/java.html
Date	(Feb 23, 2011)
Files	pom (819 bytes) jar (1.8 MB) View All
Repositories	Central Geomajas Sonatype
Used By	552 artifacts
Vulnerabilities	Direct vulnerabilities: CVE-2020-26939 CVE-2020-15522

[Maven](#)
[Gradle](#)
[Gradle \(Short\)](#)
[Gradle \(Kotlin\)](#)
[SBT](#)
[Ivy](#)
[Grape](#)
[Leiningen](#)
[Buildr](#)

```
<!-- https://mvnrepository.com/artifact/org.bouncycastle/bcprov-jdk16 -->
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcprov-jdk16</artifactId>
  <version>1.46</version>
</dependency>
```

☒ Include comment with link to declaration

Compile Dependencies (0)

Category/License	Group / Artifact	Version	Updates				
Licenses <table border="1"> <thead> <tr> <th>License</th> <th>URL</th> </tr> </thead> <tbody> <tr> <td>Bouncy Castle Licence</td> <td>http://www.bouncycastle.org/licence.html</td> </tr> </tbody> </table>				License	URL	Bouncy Castle Licence	http://www.bouncycastle.org/licence.html
License	URL						
Bouncy Castle Licence	http://www.bouncycastle.org/licence.html						

Y ahora observad: en la pestaña Maven el buscador muestra un texto XML. Ese es el texto que hay que copiar tal cual en el archivo POM para que Maven interprete que para construir el programa tiene que incluirse el archivo bcprob-jdk16.1.46.jar. Observad también que en el XML se nos muestran varias etiquetas:

groupId: Su valor (org.bouncycastle) es el nombre de un package, que se corresponde con el directorio org/bouncycastle de los repositorios central y local.

artifactId: su valor (bcprov-jdk16) es el nombre del archivo JAR sin información de versión.

versión: su valor (1.46) es la versión de la herramienta incluida en el JAR. En el caso que nos ocupa pues, el nombre local del JAR es **bcprov-jdk16.1.46.jar**.

Cuando desde NetBeans ordenéis “Clean and Build” por primera vez, Maven miraría si tiene ese jar en el repositorio local, y en caso de no tenerlo, se conectaría al repositorio remoto, lo descargaría, lo instalaría en el repositorio local y lo haría participar en la construcción del proyecto. Sucesivas operaciones de “Clean and Build” del proyecto harían que Maven recuperase el módulo jar del repositorio local.

Cuando Maven descargue el archivo y lo deposite en el repositorio local, lo hará (para un Mac) en

.m2/repository/org/bouncycastle/bcprov-jdk16/1.46/bcprov-jdk16.1.46.jar