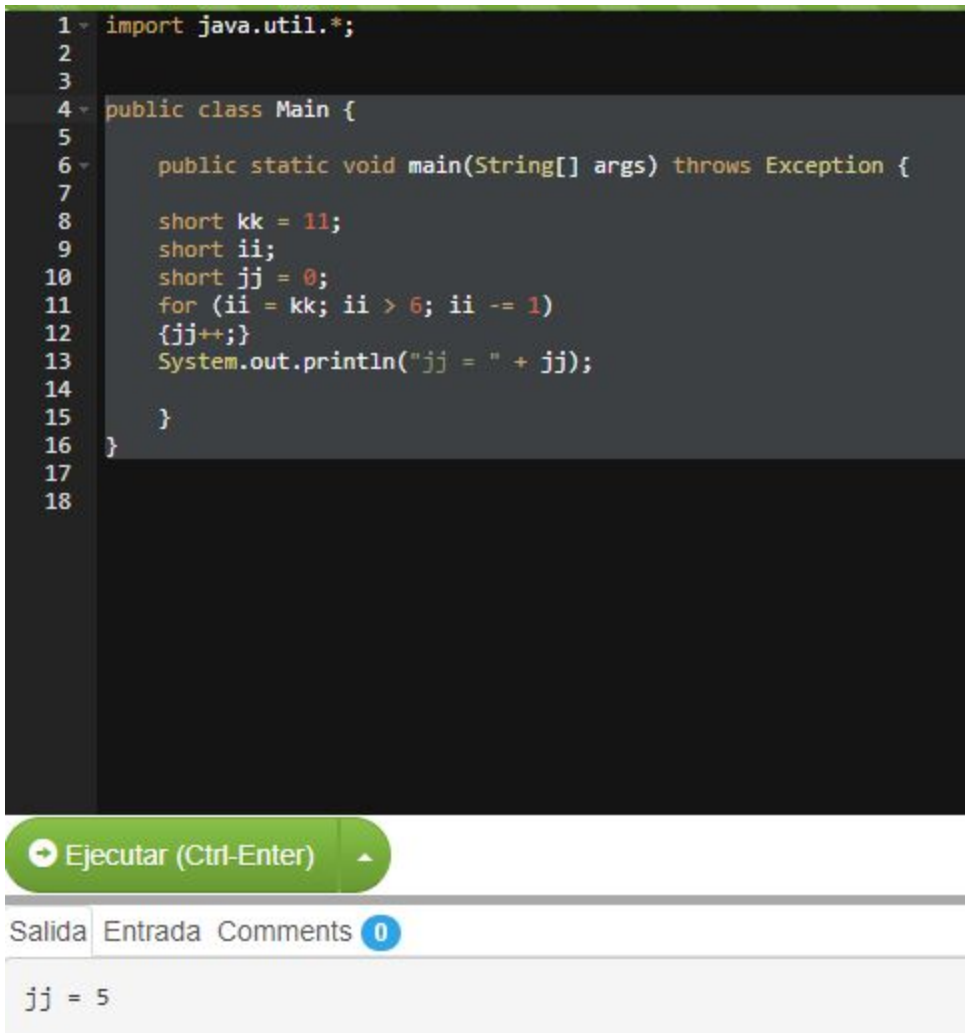
```
public class Main {

   public static void main(String[] args) throws Exception {

                        short kk = 11;
            short ii;
            short jj = 0;
            for (ii = kk; ii > 6; ii -= 1)
            {jj++;}
            System.out.println("jj = " + jj);

   }
}
```

Que valor debe tener kk para imprimir 5
short kk = 5;

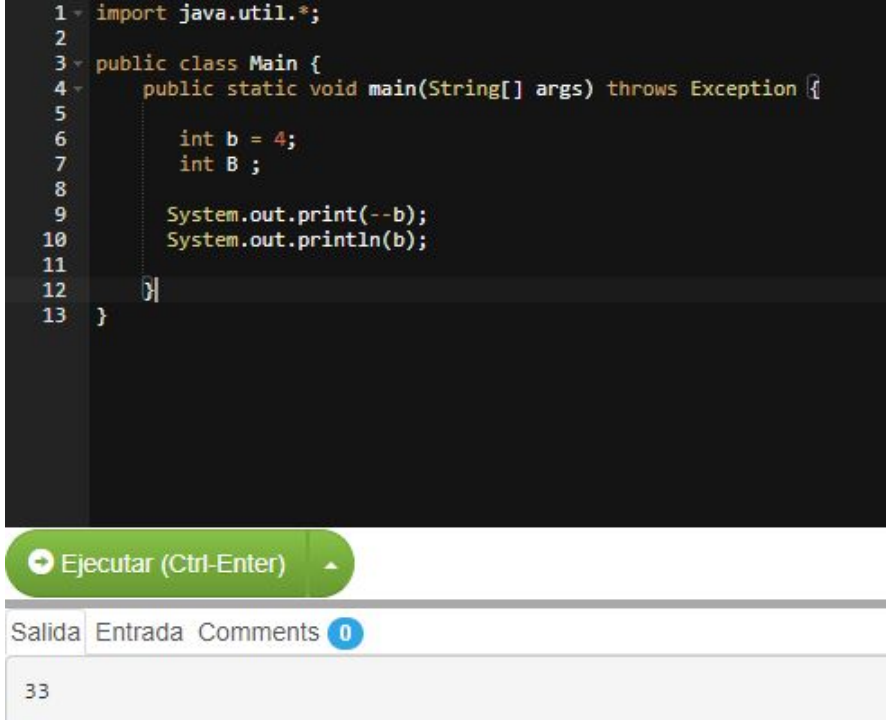


```java
import java.util.*;

public class Main {
   public static void main(String[] args) throws Exception {
      // Your code here!

     metodo();
     System.out.println("R");
      System.out.println("R" + i);

   }

   public void metodo(){
       System.out.println("R" + i);
   }


   static int i;
}
```

Que línea esta el error, public void método debe ser estático public static void

```java
import java.util.*;

public class Main {
   public static void main(String[] args) throws Exception {

      int b = 4;
      int B ;

     System.out.print(--b);
     System.out.println(b);

   }
}
```

Valor que se imprime = 33

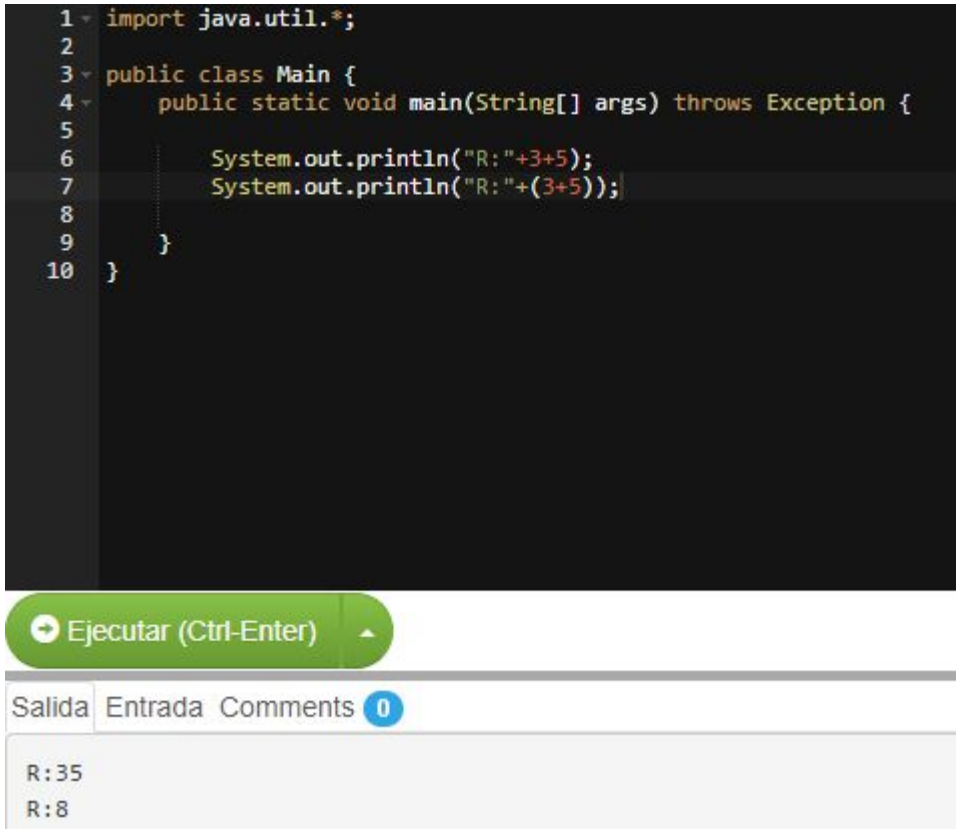



```java
import java.util.*;

public class Main {
   public static void main(String[] args) throws Exception {

      int b = 4;
      int B ;

     System.out.print(b--);
     System.out.println(b);

   }
}
```

Valor que se imprime = 43

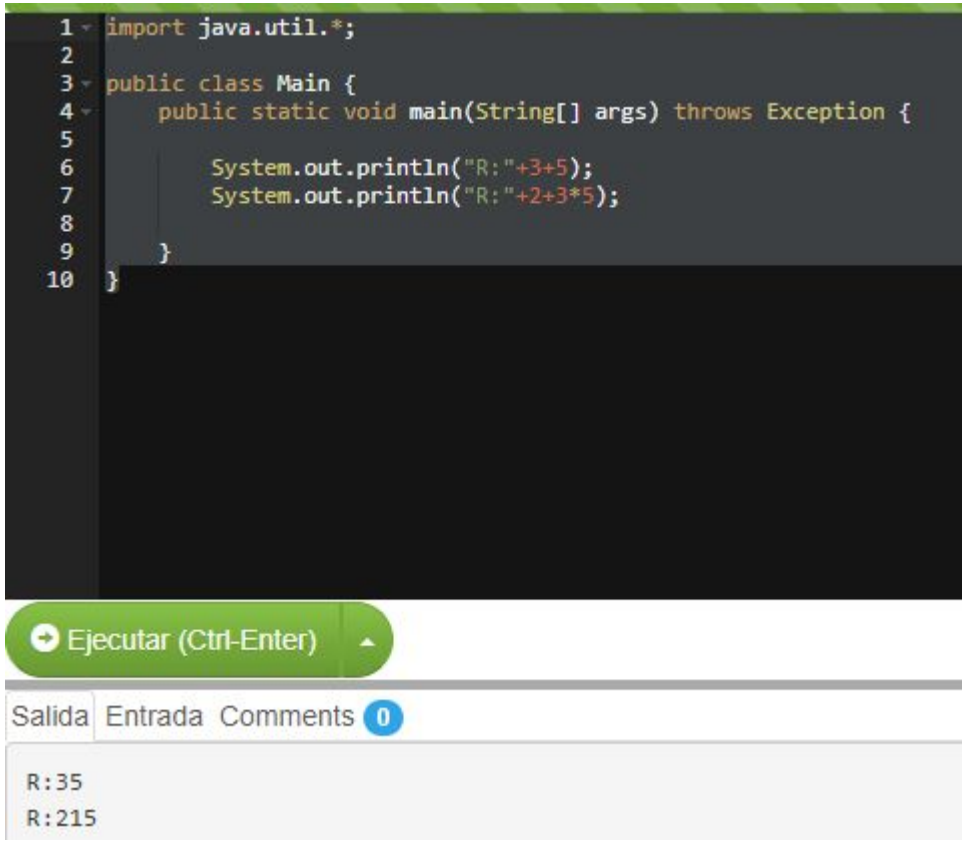
```java
import java.util.*;

public class Main {
   public static void main(String[] args) throws Exception {

      System.out.println("R:"+3+5);
      System.out.println("R:"+(3+5));

   }
}
```

Valor que se imprime = 35 y 8



```java
import java.util.*;

public class Main {
   public static void main(String[] args) throws Exception {

      System.out.println("R:"+3+5);
      System.out.println("R:"+2+3*5);

   }
}
```

Valor que se imprime = 35 y 15



```
import java.util.*;

public class Main {
   public static void main(String[] args) throws Exception {

      ClassA A = new ClassA();
      ClassA B = new ClassB();
      ClassA C = new ClassC();

      System.out.println(A);
      System.out.println(B);
      System.out.println(C);

   }

   public static class ClassA {
            public ClassA() {
              System.out.println("ClassA");        }
   }

   public static class ClassB extends ClassA{
            public ClassB() {
              System.out.println("ClassB");
            }
   }

   public static class ClassC extends ClassA{
            public ClassC() {
              System.out.println("ClassC");
            }
   }

}
```

Que se imprimira

```
1  import java.util.*;
2
3
4  public class Main {
5      public static void main(String[] args) throws Exception {
6
7          ClassA A = new ClassA();
8          ClassA B = new ClassB();
9          ClassA C = new ClassC();
10
11         System.out.println(A);
12         System.out.println(B);
13         System.out.println(C);
14
15     }
16
17     public static class ClassA {
18         public ClassA() {
19
20           System.out.println("ClassA");
21
22         }
23     }
24
25     public static class ClassB extends ClassA{
26         public ClassB() {
27
28           System.out.println("ClassB");
29
30         }
31     }
32
33     public static class ClassC extends ClassA{
34         public ClassC() {
35
36           System.out.println("ClassC");
37
38         }
39     }
40
41 }
```

Ejecutar (Ctrl-Enter)

Salida | Entrada | Comments 0

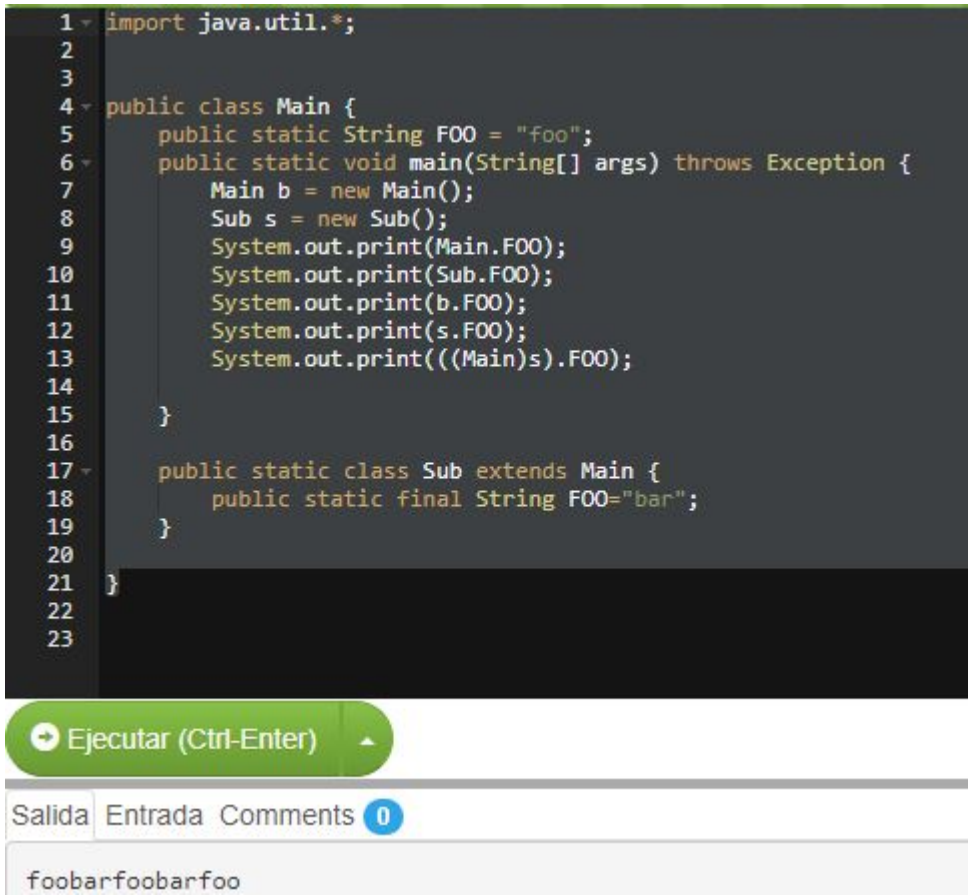
```
ClassA
ClassA
ClassB
ClassA
ClassC
```

```
import java.util.*;


public class Main {
   public static void main(String[] args) throws Exception {

      ClassA A = new ClassA();
      ClassA B = A;
      ClassA C = B;

      System.out.println(C);

   }

   public static class ClassA {
          public ClassA() {

            System.out.println("ClassA");

          }
   }
}
```

Objeto new classA y referencia es A

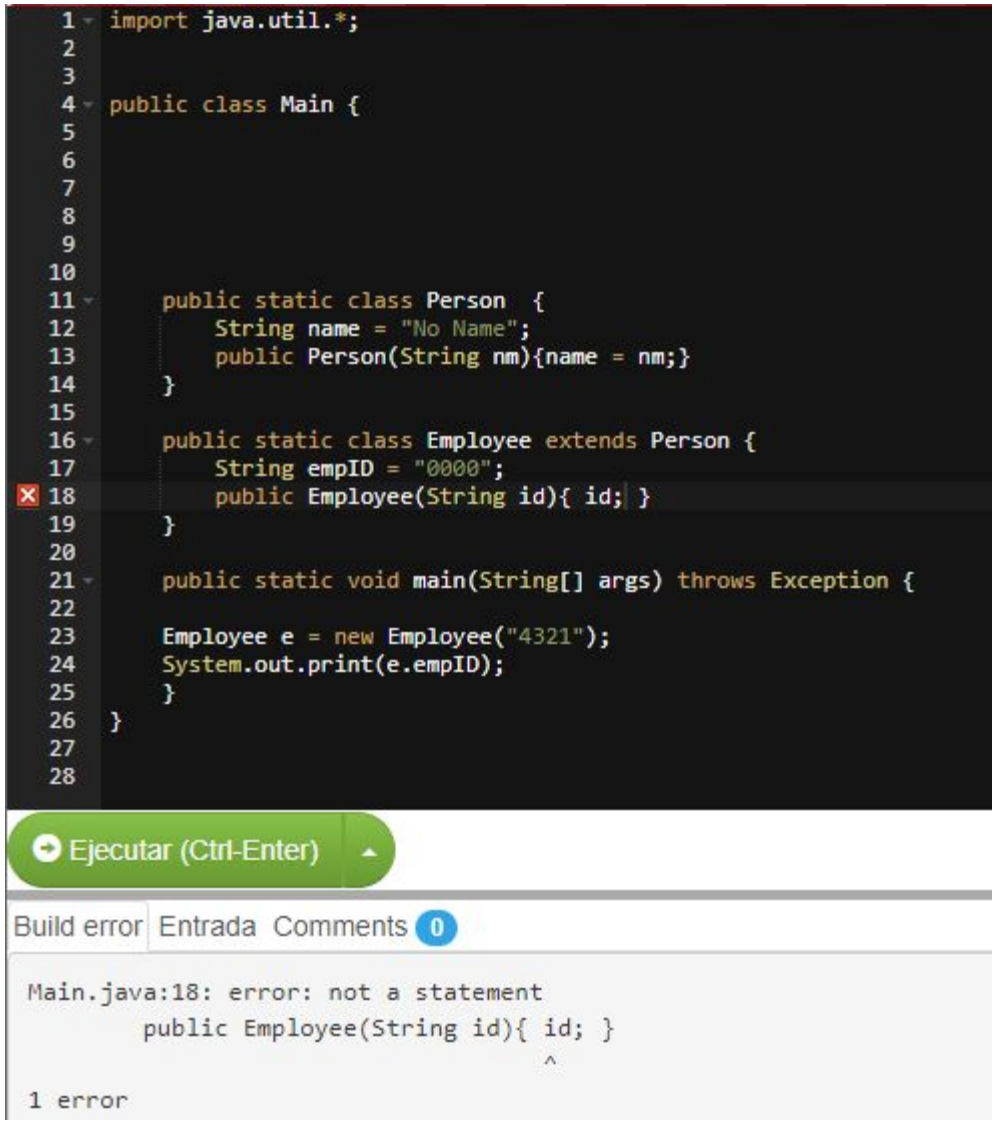
```
import java.util.*;


public class Main {
   public static String FOO = "foo";
   public static void main(String[] args) throws Exception {
      Main b = new Main();
      Sub s = new Sub();
      System.out.print(Main.FOO);
      System.out.print(Sub.FOO);
      System.out.print(b.FOO);
      System.out.print(s.FOO);
      System.out.print(((Main)s).FOO);

   }

   public static class Sub extends Main {
      public static final String FOO="bar";
   }

}
```

Que debe imprimirse:
foobarfoobarfoo



```
public class Main {

   public static void main(String[] args) throws Exception {

   Boolean b1 = true;

   Boolean b2 = false;

   int i = 0;

   //A. b1.compareTo(b2)

   //B. i = 1

   //C. i == 2? -1 : 0

   //D. "foo".equals("bar")

   while ( "foo".equals("bar") ) { System.out.print("R"); }

   }

}
```

```
import java.util.*;


public class Main {




   public static class Person  {
      String name = "No Name";
      public Person(String nm){name = nm;}
   }

   public static class Employee extends Person {
      String empID = "0000";
      public Employee(String id){ empID= id; }
   }

   public static void main(String[] args) throws Exception {

   Employee e = new Employee("4321");
   System.out.print(e.empID);
   }
}
```



¿cuál es el output?

```
import java.util.*;

class X{
   public X() { System.out.println(1); }
   public X(int numero) { System.out.println(2); }
}

class Y extends X{
   public Y() { System.out.println(3);}
   public Y(int numero) { System.out.println(4); }
}
public class Main {
   public static void main(String[] args) throws Exception {
      Y y =new Y(5);
   }
}
```

```
1  import java.util.*;
2
3  class X{
4      public X() { System.out.println(1); }
5      public X(int numero) { System.out.println(2); }
6  }
7
8  class Y extends X{
9      public Y() { System.out.println(3);}
10     public Y(int numero) { System.out.println(4); }
11 }
12 public class Main {
13     public static void main(String[] args) throws Exception {
14         Y y =new Y(5);
15     }
16 }
```

Ejecutar (Ctrl-Enter)

Salida Entrada Comments 0

```
1
4
```

¿En qué línea falla?

```
import java.util.*;

public class Main {
   public static void main(String[] args) throws Exception {
      int x1 = x2;
      int x2 = 4;
      ++j;
      metodo();
   }
   static int j=9;

   public static void metodo (){
      System.out.println(j);
   }
}
```

```
1  import java.util.*;
2
3  public class Main {
4      public static void main(String[] args) throws Exception {
5          int x1 = x2;
6          int x2 = 4;
7          ++j;
8          metodo();
9      }
10     static int j=9;
11
12     public static void metodo (){
13         System.out.println(j);
14     }
15 }
```

Ejecutar (Ctrl-Enter)

Build error | Entrada | Comments 0

```
Main.java:5: error: cannot find symbol
        int x1 = x2;
                 ^
  symbol:   variable x2
  location: class Main
1 error
```

¿cuál es el output?

```
import java.util.*;

class Employee{
   String name ="";
   int age = 18;
}
class Bob extends Employee{}
class Julian extends Employee{}

public class Main {
   public static void main(String[] args) throws Exception {
      Bob bob = new Bob();
      bob.name ="Bob";
      Julian julian = new Julian();
      julian.name = "Julian";

      System.out.println("Bob's name: "+bob.name);
   }
}
```

```
1  import java.util.*;
2
3  class Employee{
4      String name ="";
5      int age = 18;
6  }
7  class Bob extends Employee{}
8  class Julian extends Employee{}
9
10 public class Main {
11     public static void main(String[] args) throws Exception {
12         Bob bob = new Bob();
13         bob.name ="Bob";
14         Julian julian = new Julian();
15         julian.name = "Julian";
16
17         System.out.println("Bob's name: "+bob.name);
18     }
19 }
```

Ejecutar (Ctrl-Enter)

Salida Entrada Comments 0

```
Bob's name: Bob
```

¿cuál es la salida?

```
import java.util.*;

public class Main {
   public static void main(String[] args) throws Exception {
      String x="Hello world";
      char a = x.charAt(11);
      System.out.println(a);
   }
}
```

DTO

La arquitectura APX soporta el patrón Data Transfer Object, tanto en para entorno Online, como para el entorno Batch. El uso de este patrón es de uso obligado siempre que:
Se necesite una agrupación de datos simples en una clase con cierta funcionalidad asociada.
Se quiera intercambiar información entre diferentes componentes en APX de forma coherente, organizada y agrupada.
Implementar un Servicio Backend
Como características de uso, cabe destacar:
Es aplicable para las siguientes combinaciones:

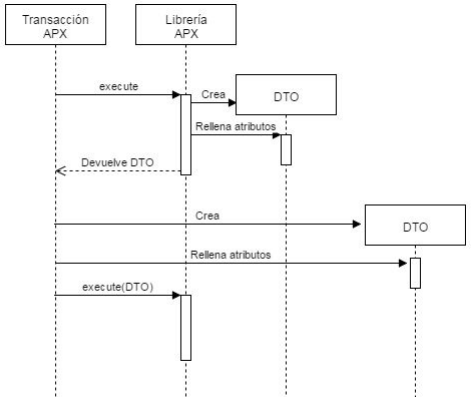
Transacción⟷ Librería

Librería ⟷ Librería

Job ⟷ Librería

Se pueden generar jerarquías de clases DTO, con la única restricción de que estas clases no tengan dependencias circulares entre ellas.
Las transacciones y las librerías pueden recibir como parámetro de entrada y de salida DTOs.
El uso de este patrón ayuda a identificar las agrupaciones de datos simples y/o complejos, y evita duplicidad de código, ya que las diferentes clases creadas, pueden ser potencialmente utilizables dentro de la misma aplicación o UUAA (o por otras UUAAs).

Es obligatorio que los DTOs se implementen dentro de un empaquetado independiente (Bundle DTO) de los componentes de APX de más alto nivel (Transacciones, Librerías y Batch). **Esta directriz es de obligado cumplimiento**, ya que si no el patrón carece de sentido.



Steps en un Batch

Tasklet y Chunk

El componente tasklet de tipo cadena es el utilizado cuando queremos realizar un flujo de procesamiento aplicado a una lista de registros (Reader, Processor y Writter por cada registro).

Las propiedades que se muestran al seleccionar un nodo tasklet son las mismas que las que se muestran al seleccionar el nodo chunk ya que en los tasklet cadena se trata como una entidad lógica completa.

commit-interval.- Intervalo de elementos procesados antes de hacer commit
reader.- Indica el bean que define el lector de este step
processor.- Indica el bean que define el procesamiento de este step
writer.- Indica el bean que define el escritor de este step
skip-limit.- Máximo número de elementos que pueden saltarse en el procesamiento
skip-policy.- Politica de salto de elementos
retry-policy.- Politica de reintento de elementos
retry-limit.- Número máximo de reintentos para procesar un elemento
chunk-completion-policy.- Politica de completitud del chunk
cache-capacity.- Capacidad de la cache para la política de reintentos
reader-transactional-queue.- Indica cuando el lector es una cola transaccional
processor-transactional.- Indica cuando el procesamiento tiene que ser transaccional

reader.- Permite definir el bean del reader en este mismo sitio en lugar de usar una referencia.
proccesor.- Permite definir el bean del procesador en este mismo sitio en lugar de usar una referencia
writer.- Permite definir el bean del writer en este mismo sitio en lugar de usar una referencia
skip-policy.- Permite definir el bean de la politica de salto en este mismo sitio en lugar de usar una referencia
retry-policy.- Permite definir el bean de la política de reintento en este mismo sitio en lugar de usar una referencia
skippable-exception-classes.- Permite definir el listado de excepciones que pueden saltarse o no deben saltarse. Permiten los siguientes subelementos:
include.- define un elemento a incluir en la lista. Tendrá el siguiente atributo:

§ class.- Indica la clase que debe excluirse
exclude.- define un elemento a excluir de la lista. Tendrá el siguiente atributo:

§ class.- Indica la clase que debe excluirse
retryable-exception-classes.- Permite definir el listado de excepciones que pueden reintentarse o no. Tiene los mismos subelementos que skippable-exception-classes
listeners.- Incluirá los listeners que contiene este chunck. Estos listeners son del mismo tipo que los listener del step

Anti patrón Blob

Este patrón suele representarse como una única librería (a partir de ahora “The Blob”) que contiene la mayoría de la funcionalidad y que otras librerías utilizan incorrectamente de apoyo.

La clave del problema es que “The Blob” puede contener un porcentaje muy elevado de responsabilidades dentro de la solución a diseñar.
Una librería compuesta de sola clase (“The Blob”) con gran cantidad de métodos “execute”.
”The Blob” normalmente es muy complicada de reutilizar y probar. Puede ser ineficiente o introducir excesiva complejidad si se reutiliza para pequeñas partes de su funcionalidad.
“The Blob” puede ser costosa de cargar en memoria, usando recursos excesivos incluso para operaciones simples.
Una modificación de “The blob” puede ser crítica para múltiples operativas para librerías dependientes. El volumen de pruebas de regresión puede ser muy alto.
El despliegue de “The Blob” causa que todas las librerías relacionadas deban reinstalarse, creando un problema de indisponibilidad temporal del servicio y una carga adicional al sistema.

Causas
Reparto de responsabilidades incorrecto. Es conveniente que se realice un diseño correcto para las nuevas librerías.
Falta de cumplimiento de la arquitectura. A veces, este anti-patrón crece accidentalmente, incluso en una arquitectura bien planteada. Esto puede ser el resultado de una inadecuada revisión del diseño de las librerías antes de su desarrollo.
Intervención demasiado limitada. En proyectos iterativos, los desarrolladores tienden a agregar pequeñas piezas de funcionalidad a clases existentes (“The Blob”) que ya están funcionando, en lugar de agregar nuevas librerías o revisar las librerías disponibles para una ubicación más eficaz de las responsabilidades.

Soluciones

La solución pasa por realizar un “refactoring” del rediseño entre las librerías involucradas.
La clave es mover comportamientos de “The Blob” a sus librerías relacionadas, diferenciando por las entidades con las que trabajan, reduciendo el acoplamiento entre librerías y simplificando los mantenimientos.

Conclusion
Una distribución de responsabilidades entre librerías adecuada causa un menor acoplamiento, simplificando cada una de las partes y disminuyendo el impacto negativo que causa “The Blob” en su ciclo de implantación y ejecución.
El factor más importante es que, en la mayoría de los casos, es mucho menos costoso crear un diseño apropiado, que volverlo a implementar correctamente. Invertir tiempo por adelantado en un buen diseño, puede asegurar que un diseño no acabe siendo un “The Blob” y/o muchos otros anti-patrones.

Contenedor Magico

Este patrón suele representarse en una librería que contiene un solo método “execute” con parámetros que actúan como filtros y que engloba diversas funcionalidades dispares en función de sus parámetros de entrada.

Otros posibles síntomas/consecuencias serían:

Es muy complicada de probar. Dado que contiene múltiples funcionalidades.
Alta complejidad ciclomática, ya que son necesarias muchas sentencias de control para realizar correctamente el filtrado.
Una modificación del método “execute” puede ser crítica para múltiples operativas para librerías dependientes. El volumen de pruebas de regresión puede ser muy alto.
Causas

Reparto de responsabilidades incorrecto. Es conveniente que se realice un diseño correcto para las nuevas librerías (apartado 5).
Falta de cumplimiento de la arquitectura. A veces, este anti-patrón crece accidentalmente, incluso en una arquitectura bien planteada. Esto puede ser el resultado de una inadecuada revisión del diseño de las librerías antes de su desarrollo.
Intervención demasiado limitada. En proyectos iterativos, los desarrolladores tienden a agregar pequeñas piezas de funcionalidad al método con nuevos filtros que ya está funcionando, en lugar de agregar nuevas librerías o revisar las librerías disponibles para una ubicación más eficaz de las responsabilidades.
Alto acoplamiento entre librerías. El uso creciente de esta librería causa un único punto de error para múltiples librerías. Una modificación incorrecta causaría la indisponibilidad de múltiples librerías dependientes.

Soluciones

La solución pasa por realizar un “refactoring” de la librería y una disgregación por entidades.

Existen dos variantes en función del problema:

Un método “execute” que contiene múltiples funcionalidades sobre la misma entidad.
En este caso la solución parte por exponer varios métodos “executeXXX”, cada uno con la lógica exclusiva de su operativa.

Un método “execute” que contiene múltiples funcionalidades sobre varias entidades. Este caso es más complejo, y su solución pasa por realizar lo siguiente:

1. Enumerar las diferentes operativas y agruparlas por entidad.
2. Realizar los pasos del apartado 1 para la entidad sobre las que la librería debería trabajar.
3. Extraer las funcionalidades pertenecientes a otras entidades a una o varias librerías específicas por cada entidad.

Normalmente es posible detectarlo a posteriori en procesos de integración continua, donde uno de los pasos es analizar métricas de código, no obstante lo ideal es anticiparse realizando una separación de funcionalidades de forma organizada por entidades.

Una distribución de responsabilidades entre librerías adecuada causa un menor acoplamiento, simplificando cada una de las partes y disminuyendo el impacto negativo que causa el “Contenedor Mágico” en su ciclo de implantación y ejecución.

El factor más importante es que, en la mayoría de los casos, es mucho menos costoso crear un diseño apropiado, que volverlo a implementar correctamente. Invertir tiempo por adelantado en un buen diseño, puede asegurar que un diseño no acabe siendo un “Contenedor Mágico” y/o muchos otros anti-patrones.

Loggers

Los niveles más comunes son **DEBUG, INFO, WARNING y ERROR**. La clasificación de los diferentes eventos en cada nivel forma parte del ejercicio de análisis, y debe estar orientada a la traza legible y útil:
DEBUG, para información de muy bajo nivel solo útil para depurar la aplicación, tanto en el desarrollo como en el análisis de incidencias:
Llamadas a funciones y procedimientos y otros componentes, con parámetros y respuestas.
Flujos de ejecución.
Desarrollo de algoritmos y procedimientos que permitan identificar y seguir su ejecución en desarrollo.
INFO, información de nivel superior que permite monitorear la ejecución normal:
Paradas y arranques de servicios y sistemas.
Parámetros de configuración críticos o relevantes.
Inicio y finalización de transacciones y operaciones completas.
Cambios del estado de operaciones.
WARN, información de situaciones, que aun sin ser de error, si son anómalas o no previstas, aunque la aplicación tiene alternativas a resolver:
Parámetros no definidos y cuyo valor se toma por defecto.
Situaciones anómalas, pero que son resueltas por la aplicación, dejando el funcionamiento en un estado correcto.
Funciones no imprescindibles ni imprescindibles, que no se pueden solucionar, pero dejan el funcionamiento en un estado correcto.
ERROR, información de situaciones que son error y que impiden la correcta ejecución de una operación o transacción, pero sin afectar otras operaciones o transacciones (error aislado o contenido):
No se pudo realizar una operación o transacción, pero no afecta a otras solicitudes o consultas erróneas (almacenando los parámetros de entrada).
Funciones Aspectos generales de la aplicación, que incluso afectando al funcionamiento general de la aplicación, no se consideran imprescindibles ni imprescindibles.
Recordamos que es una mala práctica capturar excepciones genéricas y está totalmente prohibido su tratamiento (ver Tratamiento de Excepciones).

Paginacion en transacciones

Se utilizará este patrón siempre que se requiera dividir un conjunto de datos en varios subconjuntos o páginas y adicionalmente se necesite acceder a ellas de forma aislada.

El uso de este patrón es conveniente siempre para mejorar la eficiencia de consultas paginadas a orígenes de datos variados, obteniendo únicamente lo que se necesita.

Este tipo de paginación es el que establece Arquitectura y Calidad como el más eficiente, tanto en consumo de recursos como en el tiempo de proceso para la consecución de la información, lo que redunda en un menor tiempo de espera por parte del usuario final.

Las aplicaciones deben calcular el tamaño óptimo que debe contener el bloque del mensaje de respuesta basándose en los siguientes factores:

Número de filas presentadas por página
Número máximo de páginas que pueden ser enviadas dentro del mensaje de envío
El mensaje de entrada que se recibe desde la aplicación debe incluir un tipo DTO (paginationIn) con los siguientes atributos:

Clave de paginación o reposicionamiento (paginationKey). Únicamente se informa en posteriores llamadas. El tamaño máximo de este campo para la transacción será el que defina el propio aplicativo.
Tamaño de página (pageSize)

El mensaje de salida, además de la información propia del servicio solicitado, debe incluir un tipo de dato DTO (paginationOut) con los siguientes atributos obligatorios:

Indicador de si hay o no más datos por recuperar, que informará al aplicativo sobre la necesidad o no de ejecutar otra transacción para obtener el resto de información (hasMoreData)
La clave de reposicionamiento con la que deben volver a invocar la transacción para recuperar más filas (paginationKey). El tamaño máximo de este campo para la transacción será el que defina el propio aplicativo.

Las llamadas a las librerías de paginación deben incluir el mismo DTO de entrada definido por la clase PaginationInDTO (implementado en un componente de tipo DTO) que el que se define anteriormente para la transacción como campo “paginationIn”. La librería por su parte devolverá la página encontrada (si hay elementos que la compongan), y a su vez el DTO definido en la clase PaginationOutDTO (implementad en un componente de tipo DTO), definido para el campo de salida de la transacción “paginationOut”.

PAGINACIÓN EN LIBRERÍAS APX

Se utilizará este patrón siempre que se requiera dividir un conjunto de datos en varios subconjuntos o páginas y adicionalmente se necesite acceder a ellas de forma aislada.

El uso de este patrón es conveniente siempre para mejorar la eficiencia de consultas paginadas a orígenes de datos variados, obteniendo únicamente lo que se necesita.

Este tipo de paginación es el que establece Arquitectura y Calidad como el más eficiente, tanto en consumo de recursos como en el tiempo de proceso para la consecución de la información, lo que redunda en un menor tiempo de espera por parte del usuario final.

Las aplicaciones deben calcular el tamaño óptimo que debe contener el bloque del mensaje de respuesta basándose en los siguientes factores:

Número de filas presentadas por página
Número máximo de páginas que pueden ser enviadas dentro del mensaje de envío
El mensaje de entrada que se recibe desde la aplicación debe incluir el DTO de la clase PaginationInDTO (es necesario implementarlo) con los siguientes atributos:

Clave de paginación o reposicionamiento (paginationKey). Únicamente se informa en posteriores llamadas
Tamaño de página (pageSize)

El mensaje de salida, además de la información propia del servicio solicitado, debe incluir el DTO de la clase PaginationOutDTO (es necesario implementarlo) con los siguientes atributos:

Indicador de si hay o no más datos por recuperar, que informará al aplicativo sobre la necesidad o no de ejecutar otra consulta para obtener el resto de información (hasMoreData)
La clave de reposicionamiento con la que deben volver a invocar la librería para recuperar más filas (paginationKey)

El área de entrada es particular y conocida por cada aplicación y APX no posee ningún dato relativo a paginación.

Según este contenido la librería **deberá componer las consultas** utilizando las librerías proporcionadas en APX y acceder a los diferentes Orígenes de Datos para obtener la información solicitada.

PATRÓN CRUD EN LIBRERÍAS APX

Se utilizará este patrón siempre que se requieran algunas de las operaciones básicas (Create, Read, Update y Delete) sobre una entidad.

Este patrón se aplica a cualquier medio de almacenamiento (BBDD, archivos, etc).

El uso de este patrón ayuda a tener un mejor mantenimiento de las operativas por entidades y sobre todo para mejorar la reutilización de librerías (mayor granularidad).

Otra característica es la centralización en una librería de todas las operaciones básicas sobre una entidad.

Existen dos aproximaciones:

Parcial: solo se implementan algunas de las operaciones básicas (por ejemplo, lecturas de tablas maestras de BBDD, siempre que se añadan nuevas entradas mediante scripts DML).
Completa: se implementan todas las operaciones básicas.

Cada operación debe tener una implementación ligera, evitando extender la lógica más allá del objetivo de la operación (por ejemplo, mezclando operativas con otras entidades).

Como contraindicación, **no se deben tener varias librerías parciales para una única entidad**, ya que estaríamos desvirtuando el patrón. Si es posible (y recomendable), mantener todas las operaciones sobre la entidad encapsuladas en la librería, como lecturas avanzadas, etc.

Excepciones - Manejo de excepciones

El código de la aplicación no puede manejar las excepciones lanzadas por los conectores proporcionados por APX, por los productos invocados por terceros, ni ninguna excepción que herede de RuntimeException.
Por lo tanto, el código try ... catch (Exception exc) no está permitido.
Si ocurre una excepción, la Arquitectura la capturará y habilitará la gestión de errores.
También será responsable de deshacer aquellos accesos donde exista transaccionalidad (Acceso a bases de datos, etc ...).
Las únicas excepciones que pueden ser capturadas y administradas por las aplicaciones son las siguientes:
Excepciones de la arquitectura APX:

- com.bbva.apx.exception.business.BusinessException
- com.bbva.apx.exception.io.network.TimeoutException
- com.bbva.apx. exception.db.DuplicateKeyException
- com.bbva.apx.exception.db.IncorrectResultSizeException
- com.bbva.apx.exception.db.NoResultException
- com.bbva.apx.exception.db.TimeoutException
- com.bbva.elara.utility.interbackend.cics.exceptions.BusinessException

Excepciones de Java:

- java.lang.NumberFormatException

Excepciones de Spring :

- org.springframework.web.client.RestClientException

Excepciones de aplicación.

Invocaciones a transacciones MainFrame

Es deseable que el código APX de la transacción NO invoque ninguna otra transacción Mainframe para evitar acoplamientos y dependencias entre ellos.
Siempre que sea posible, debe 'replicar / copiar' los datos de lectura en la base de datos de la aplicación en APX.

Si la transacción APX invoca MainFrame, debe invocar un máximo de dos veces y si debe hacerlo más de una vez debe tener la validación del Arquitecto / Solucions que está involucrado en el proyecto.

Anidamiento e invocación finita de bibliotecas

El árbol de dependencia en la invocación a las bibliotecas debe ser finito, de modo que se eviten profundidades superiores a 3 niveles (Transacción> Biblioteca> Biblioteca> Biblioteca)

Así mismo se debe evitar una transacción o librería, dentro de su lógica, invocar más de 9 libreries.

Esta restricción se basa no solo en razones de legibilidad y trazabilidad del código, sino también en tratar de minimizar el impacto del hecho de que una biblioteca mostrada incorrectamente puede tener otras bibliotecas y / o transacciones. Una biblioteca que no se despliega correctamente impide que los componentes que pueden consumirla se instalen y activen correctamente. Cuanto mayor sea el número de módulos dependientes, mayor será la probabilidad de falla.

Invocación entre transacciones

No se permite la invocación sincrónica.

En caso de que una transacción deba invocar a otra, solo se puede realizar de forma asincrónica, después de ser indicada por el APX Architect asignado al proyecto.

La invocación asincrónica entre transacciones solo debe aplicarse en casos de transacciones no críticas, en las que se puede ignorar el estado de finalización de la transacción asincrónica, por lo que no es necesario esperar a la finalización de dicha transacción para continuar con la operativa.

Una transacción asincrónica no se puede volver a invocar desde otra transacción asincrónica.

La Arquitectura APX o Arquitectura Backend Java Extendida

Fue creada con el objetivo de ser una extensión de la Arquitectura Backend proporcionando las mismas capacidades en el mundo distribuido.

Basado en tecnologías de código abierto.

Ofrece una alternativa confiable para el desarrollo de transacciones que no está acoplada al canal.
Actúa como una extensión de la plataforma Mainframe.

Apoya la lógica de las aplicaciones que se adaptan a esta plataforma.

Ayuda a reducir el uso del Mainframe

Las operaciones de misión crítica para el banco se ejecutan bajo APX.

70.- Los niveles mas comunes son DEBUG, INFO, WARNING Y ERROR

a) Verdadero
b) Falso

69.- Dentro de los casos de uso de la arquitectura de APX y con el fin de estandarizar las aplicaciones del banco se plantearon dos paradigmas compatibles con BBVA:

a) BackEnd banca tradicional
b) Servicios Distribuidos Backend

66.- Menciona al menos dos restricciones que se tienen en APX en la invocación entre transacciones

a) La invocación asíncrona entre transacciones escasos de transacciones criticas
b) La invocación asíncrona entre transacciones escasos de transacciones NO criticas
c) No se permite La invocación asíncrona entre transacciones
d) No se permite La invocación síncrona entre transacciones
e) Desde una transacción asíncrona se puede invocar nuevamente a otra transacción asíncrona
f) Desde una transacción asíncrona no se puede invocar nuevamente a otra transacción asíncrona

56. ¿Qué patrones de la siguiente lista son obligatorios del Data Transfer Object (DTO) en APX?

a) Se quiera intercambiar información entre diferentes componentes en APX de forma coherente, organizada y agrupada

b) Ninguna respuesta

c) Implementar un servicio backend

d) Se necesite una agrupación de datos simples en una clase con cierta funcionalidad asociada

53. ¿Que son los patrones de diseño en APX?

a) Un patrón de diseño brinda una solución a un problema de diseño
b) Es un patrón es único para resolver cada problema por lo que no es reutilizable
c) Todas las respuestas
d) Un patrón de diseño brinda una solución a un problema de diseño
e) Un patrón de diseño brinda una solución a un problema de diseño y es único para resolver cada problema por lo que no es reutilizable
f) Es un patrón efectivo para resolver problemas similares en diferentes ocasiones por lo que es reutilizable

43. De la lista siguiente indique cuales son las invocaciones restringidas en la Arquitectura Batch

a) Invocación WebServices, servicios REST
b) Invocación WebServices, servicios REST, Uso de Pre y Post Acciones y Escritura en Registro de operaciones
c) Uso de Pre y Post Acciones
d) Escritura en el Diario Electrónico
e) Acceso a recursos externos vía HTTP
f) Invocación WebSErvices, servicios REST, Acceso a recurso externos vía HTTP y Uso de Pre y Post Acciones
g) Todas las respuestas
h) Escritura en registro de operaciones

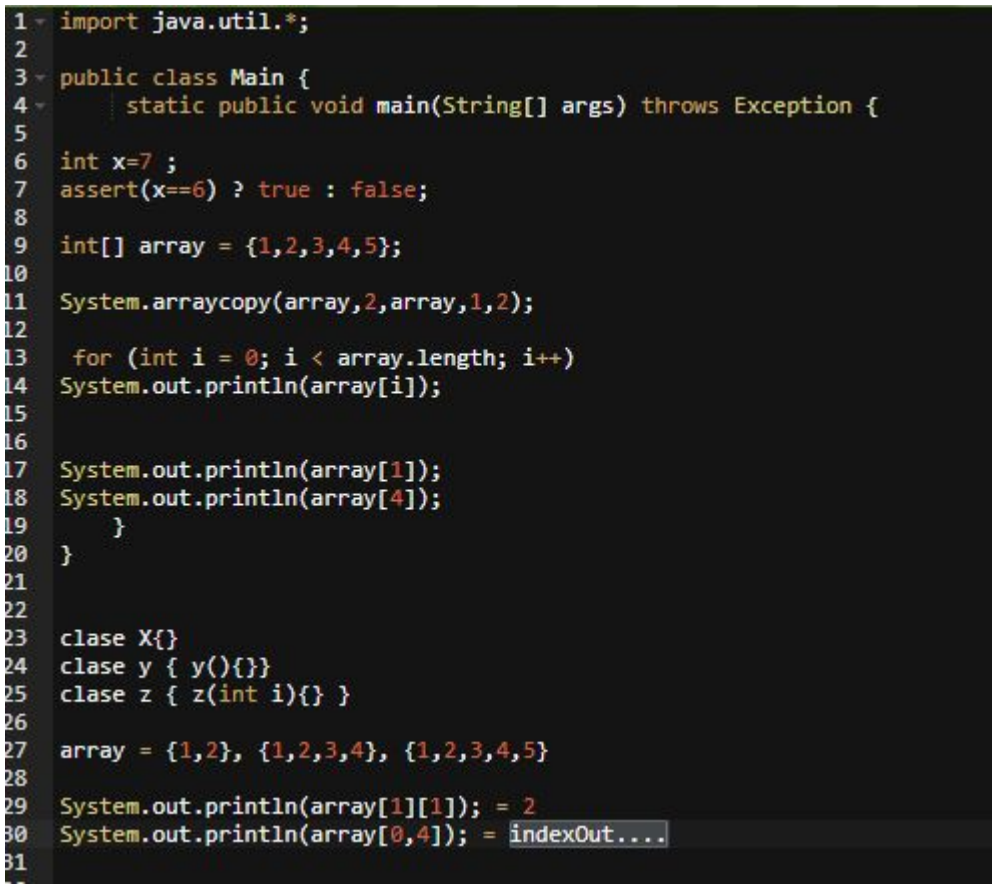
6. ¿Cuáles son las excepciones permitidas por APX?

a) com.bbva.apx.exception.business.LogicalException
com.bbva.apx.exception.io.network.TimeoutException
com.bbva.apx.exception.db.DuplicateKeyException
com.bbva.apx.exception.db.Empty
com.bbva.apx.exception.db.TimeoutException

b) com.bbva.apx.exception.business.BusinessException
com.bbva.apx.exception.io.network.TimeoutException
com.bbva.apx.exception.db.DuplicateKeyException
com.bbva.apx.exception.db.NoResultException
com.bbva.apx.exception.db.TimeoutException

c) com.bbva.apx.exception.business.LogicalException
com.bbva.apx.exception.io.network.AddressNotFound
com.bbva.apx.exception.db.DuplicateKeyException
com.bbva.apx.exception.db.Empty
com.bbva.apx.exception.db.TimeoutException

d) com.bbva.apx.exception.business.LogicalException
com.bbva.apx.exception.io.network.AddressNotFound
com.bbva.apx.exception.db.NotAllowedDataException
com.bbva.apx.exception.db.DuplicateKeyException
com.bbva.apx.exception.db.TimeoutException



```java
import java.util.*;

public class Main {
    static public void main(String[] args) throws Exception {

int x=7 ;
assert(x==6) ? true : false;

int[] array = {1,2,3,4,5};

System.arraycopy(array,2,array,1,2);

 for (int i = 0; i < array.length; i++)
System.out.println(array[i]);


System.out.println(array[1]);
System.out.println(array[4]);
    }
}


clase X{}
clase y { y(){}}
clase z { z(int i){} }

array = {1,2}, {1,2,3,4}, {1,2,3,4,5}

System.out.println(array[1][1]); = 2
System.out.println(array[0,4]); = indexOut....

```

```
import java.util.*;

public class Main {
    static public void main(String[] args) throws Exception {

int x=7 ;
assert(x==6) ? true : false;


   }
}
```

no compila

clase X{}
clase y { y(){}}
clase z { z(int i){} }

X tiene un constructor default

array = {1,2}, {1,2,3,4}, {1,2,3,4,5}

System.out.println(array[1][1]); = 2
System.out.println(array[0][4]); = indexOut....

Resultado 2 e indexOut

```
int[] array = {1,2,3,4,5};

System.arraycopy(array,2,array,1,2);

 for (int i = 0; i < array.length; i++)
System.out.println(array[i]);


System.out.println(array[1]);
System.out.println(array[4]);
```

Salida 35

```
System.arraycopy(array,2,array,1,2);
int[] array = {1,2,3,4,5};
{1,3,4,4,5};

System.arraycopy(array,2,array,1,3);
int[] array = {1,2,3,4,5};
{1,3,4,5,5};
```

StringBuilder sb=new StringBuilder("128")
StringBuilder sb=new String("128")
StringBuilder sb=new StringBuilder(128)
StringBuilder sb=new StringBuilder.setCapacity(128)

StringBuilder sb=new StringBuilder(128) con capacidad de 128

1. APX permite invocación asíncrona de transacciones

   a) Verdadero

   b) Falso

1. ¿Cuál es la función de IMS Connect?

a) Conector para BD

b) Conector para Host

c) Conector para ASO

d) Conector para Servicios de comunicacion

1. ¿Cuál es el número máximo de librerías a invocar desde una transacción?

a) 7 librerías

b) 3 librerías

c) 12 librerías

d) 9 librerías

1. ¿Cuál es la restricción al generar una jerarquía de clases DTO?

a) Que las clases no tengan dependencias circulares entre ellas

b) Que las agrupaciones no dependan entre ellas mismas

c) Que la librerias solo tengan parámetros de entrada

d) Que los patrones sean simples

Crear StringBuilder con capacidad de 128
StringBuilder sb=new StringBuilder("128")
StringBuilder sb=new String("128")
StringBuilder sb=new StringBuilder(128) CORRECTA
StringBuilder sb=new StringBuilder.setCapacity(128)

Given:
05. class Building { }
06. public class Barn extends Building {
07. public static void main(String[] args) {
08. Building build1 = new Building();
09. Barn barn1 = new Barn();
10. Barn barn2 = (Barn) build1;
11. Object obj1 = (Object) build1;
12. String str1 = (String) build1;
13. Building build2 = (Building) barn1;
14. }
15. }
Which is true?
A. If line 10 is removed, the compilation succeeds.
B. If line 11 is removed, the compilation succeeds.
C. If line 12 is removed, the compilation succeeds.
CORRECTA
D. If line 13 is removed, the compilation succeeds.
E. More than one line must be removed for compilation to succeed.

Given:

#1
```
package handy.dandy;
public class KeyStroke {
public void typeExclamation() {
System.out.println("!")
}
}
```
#2
```
package handy; /* Line 1 */
public class Greet { /* Line 2 */
public static void main(String[] args) { /* Line 3 */
String greeting = "Hello"; /* Line 4 */
System.out.print(greeting); /* Line 5 */
Keystroke stroke = new Keystroke; /* Line 6 */
stroke.typeExclamation(); /* Line 7 */
} /* Line 8 */
} /* Line 9 */
```

What three modifications, made independently, made to class greet, enable the code to compile and run?

A.Line 6 replaced with handy.dandy.keystroke stroke = new KeyStroke ( );
B.Line 6 replaced with handy.*.KeyStroke = new KeyStroke ( );
C.Line 6 replaced with handy.dandy.KeyStroke Stroke = new handy.dandy.KeyStroke(); CORRECTA
D.import handy.*; added before line 1 CORRECTA
E.import handy.dandy.*; added after line 1
F.import handy.dandy,KeyStroke; added after line 1 CORRECTA
G.import handy.dandy.KeyStroke.typeException(); added before line 1

Given:
01. public class Boxer1{
02. Integer i;
03. int x;
04. public Boxer1(int y) {
05. x = i+y;
06. System.out.println(x);
07. }
08. public static void main(String[] args) {
09. new Boxer1(new Integer(4));
10. }
11. }
What is the result?
A. Compilation fails because of an error in line 5.
B. A NumberFormatException occurs at runtime.
C. An IllegalStateException occurs at runtime.
D. The value "4" is printed at the command line.
E. Compilation fails because of an error in line 9.
F. A NullPointerException occurs at runtime. CORRECTA

Given:

String message1 = "Wham bam!";
String message2 = new String("Wham bam!");
if (message1 == message2)
System.out.println("They match");
if (message1.equals(message2))
System.out.println("They really match");
A.
They match
They really match
B.
They really match CORRECTA
C.
They match
D.
Nothing Prints
E.
They really match
They really match

```
public static void main(String[] args) {
String [] table = {"aa", "bb", "cc"};
int ii = 0;
for (String ss:table) {
while (ii < table.length) {
System.out.println (ii);
ii++;
break;
}
}
```

How many times is 2 printed?

A.zero

B.once CORRECTA

C.twice

D.thrice

E.it is not printed because compilation fails

Which two are valid declarations of a two-dimensional array?
A.int[][] array2D; CORRECTA
B.int[2][2] array2D;
C.int array2D[];
D.int[] array2D[]; CORRECTA
E.int[][] array2D[];

Given the code fragment:

```
String color = "Red";
switch(color) {
case "Red":
System.out.println("Found Red");
case "Blue":
System.out.println("Found Blue");
break;
case "White":
System.out.println("Found White");
break;
default:
System.out.println("Found Default");
}
```

What is the result?
A.
Found Red
B.
Found Red
Found Blue
C.
Found Red
Found Blue
Found White
D.
Found Red
Found Blue
Found White
Found Default

Given:

```
public class DoWhile {
public static void main (String [] args) {
int ii = 2;
do {
System.out.println (ii);
} while (--ii);
}
```

What is the result?
A.
2
1
B.
2
10
C.
null
D.
an infinite loop
E.
compilation fails CORRECTA

- Arbol de invocaciones entre transacciones y librerias : 3(TX-LIB-LIB-LIB)
- Maximo numero de invocaciones de librerias en logica: 9
- Componentes de un Tarea: Entrada, Procesamuiento, Salida
- En que se basa la arquitectura Batch de APX: SpringBatch
- Esta permitido la creacion y manipulacion de hilos en APX: falso
- Logger que refleja informacion de alto nivel, Paradas y arranques de sistemas, Parametros de config criticos, Inicio y fin de transacciones y operaciones completas: INFO
- Que NO significa APX: Todas las opciones traian la palabra ARQUITECTURA menos una que decia JAVA BACKEND PLATFORM
- Antipatron que que consta de una libreria con una unica clase con multiples metodos execute: BLOB
- Antipatron que consta de una unica clase con un solo metodo execute con parametros que actuan como filtros: Contenedor magico
- Numero maximo de invocaciones a Mainframe permitidos: 2

```
11. class Person {
12.   String name = "No name";
13.   public Person(String nm) { name = nm; }
14. }
15.
16. class Employee extends Person {
17.   String empID = "0000";
18.   public Employee(String id) { empID = id; }
19. }
20.
21. public class EmployeeTest {
22.   public static void main(String[] args) {
23.     Employee e = new Employee("4321");
24.     System.out.println(e.empID);
25.   }
26. }
```

Click the Exhibit button.
What is the result?

- A - 4321
- B - 0000
- C - An exception is thrown at runtime.
- D - Compilation fails because of an error in line 18.

Hide answers/explanations

Correct: D

Simulador Java