**Guía JAVA**

1. **Excepciones de Java permitida\***

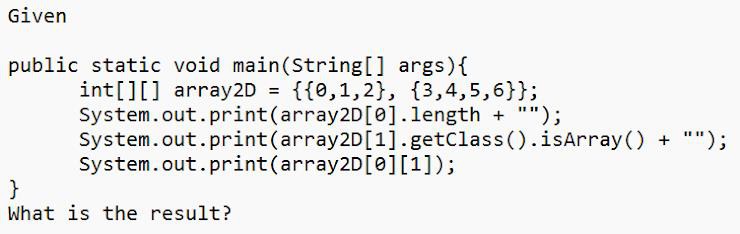
* org.springframework.web.client.RestClientException
* No conozco la respuesta
* **java.lang.NumberFormatException**
* com.bbva.apx.exception.db.NoResultException
* com.bbva.elara.utility.interbackend.cics.exceptions.BusinessException

1. **Which three are bad practices?**

* **Checking for ArrayIndexOutOfBoundsException and ensuring that the program can recover if one occurs.**
* Checking for FileNotFoundException to inform a user that a filename entered is not valid.
* **Checking for Error and, if necessary, restarting the program to ensure that users are unaware problems.**
* **Checking for ArrayIndexOutOfBoundsException when iterating through an array to determine when all elements.-have been visited.**

1. **Indica a JUnit que la propiedad que usa esta anotación es una simulación y, por lo tanto, se inicializa como tal y es susceptible de ser inyectada por** [**@InjectMocks.**](mailto:@InjectMocks)

* Mockito
* **Mock**
* Inject
* InjectMock

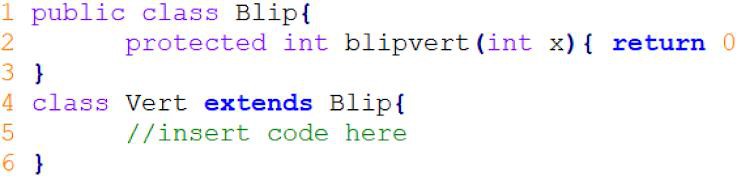


* 3false3
* 3false1
* 2false1
* **3true1**
* 2true3

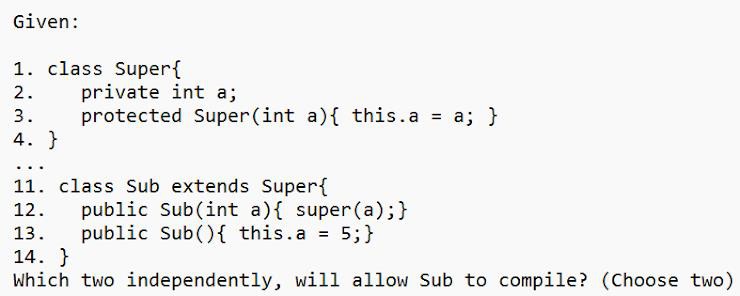
1. **Which two statments are true?**

* An interface CANNOT be extended by another interface.
* **An abstract class can be extended by a concrete class.**
* An abstract class CANNOT be extended by an abstract class. An interface can be extended by an abstract class.
* **An abstract class can implement an interface**
* An abstract class can be extended by an interface.

1. **Which five methods, inserted independiently at line 5, will compile? (Choose five)**

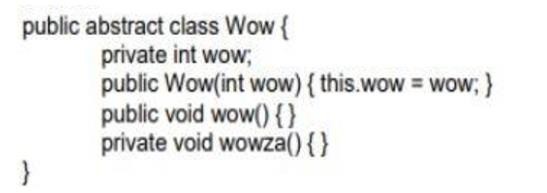


* **Private int blipvert(long x) { return 0; }**
* **Protected int blipvert(long x) { return 0; }**
* **Protected long blipvert(int x, int y) { return 0; }**
* **Public int blipvert(int x) { return 0; }**
* Private int blipvert(int x) { return 0; }
* Protected long blipvert(int x) { return 0; }
* **Protected long blipvert(long x) { return 0; }**



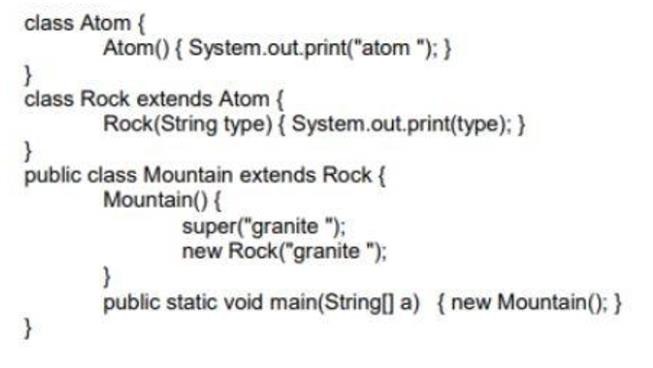
* Change line 2 to: public int a;
* **Change line 13 to: public Sub(){ super(5);}**
* Change line 2 to: protected int a;
* **Change line 13 to: public Sub(){ this(5);}**
* Change line 13 to: public Sub(){ super(a)

1. **Whats is true about the class Wow?**



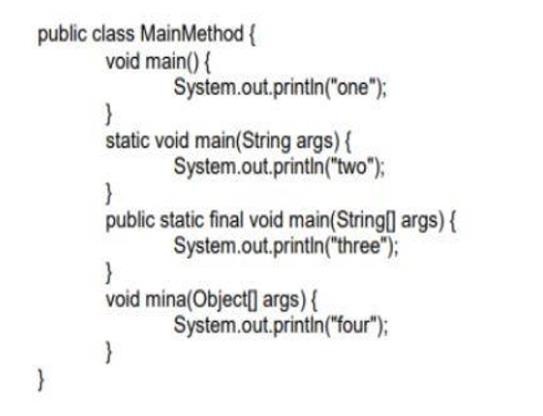
* **It compiles without error.**
* It does not compile because an abstract class cannot have private methods
* It does not compile because an abstract class cannot have instance variables.
* It does not compile because an abstract class must have at least one abstract method.
* It does not compile because an abstract class must have a constructor with no arguments.

1. **What is the result?**



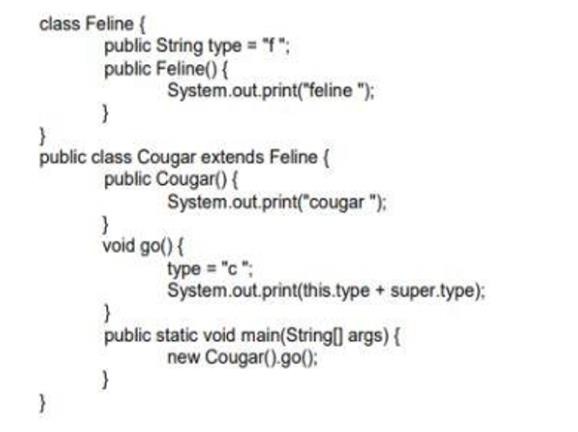
* Compilation fails.
* Atom granite.
* Granite granite.
* Atom granite granite.
* An exception is thrown at runtime
* **Atom granite atom granite**

1. **What is printed out when the program is excuted?**



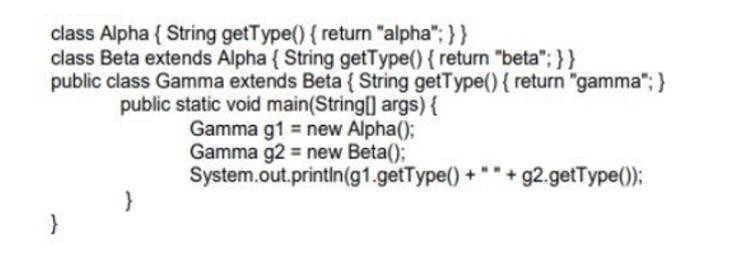
* one
* two
* **three**
* four
* There is no output

1. **What is the result?**



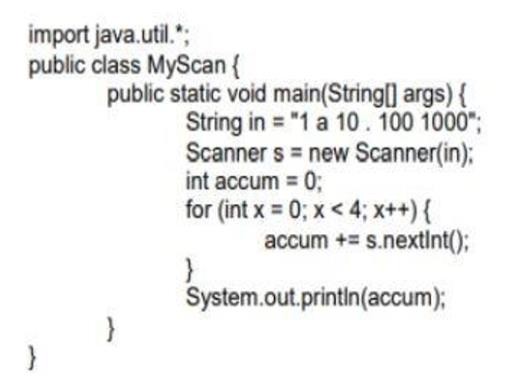
* Cougar c f.
* Feline cougar c f.
* **Feline cougar c c.**
* Compilation fails.

1. **What is the result?**



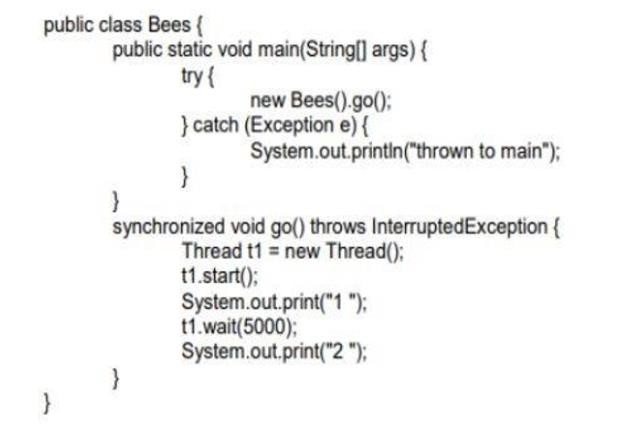
* Alpha beta
* Beta beta.
* Gamma gamma.
* **Compilation fails**.

1. **What is the result?**

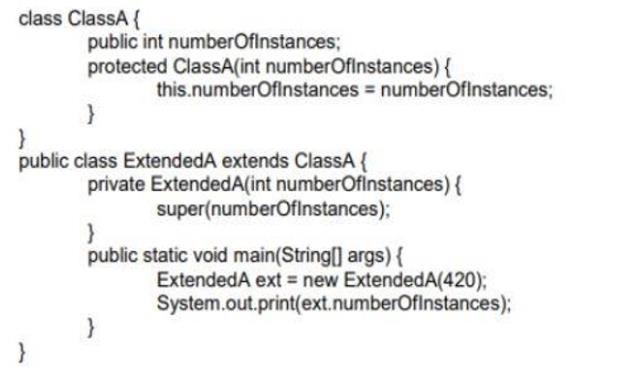


* 11
* 11
* 1111
* **An exception is thrown at runtime**

1. **What is the result?**



* The program prints 1 then 2 after 5 seconds.
* **The program prints: 1 thrown to main.**
* The program prints: 1 2 thrown to main.
* The program prints:1 then t1 waits for its notification.

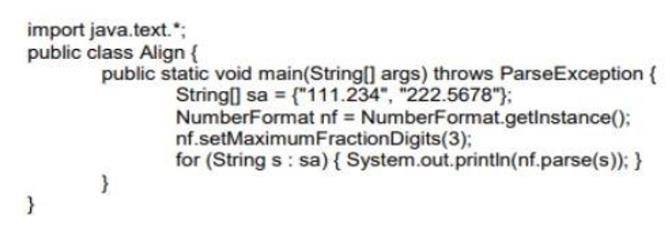
1. **Which statement is true?**

* **420 is the output.**
* An exception is thrown at runtime.
* All constructors must be declared public.
* Constructors CANNOT use the private modifier.
* Constructors CANNOT use the protected modifier

1. **The SINGLETON pattern allows:**

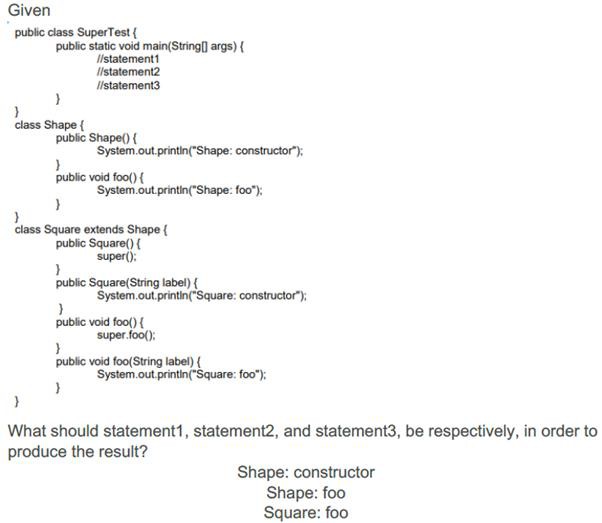
* Have a single instance of a class and this instance cannot be used by other classes
* **Having a single instance of a class, while allowing all classes have access to that instance.**
* Having a single instance of a class that can only be accessed by the first method that calls it.

1. **What is the result?**



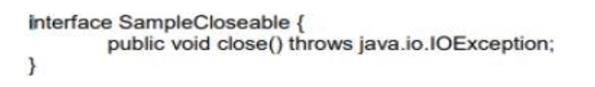
* 111.234 222.567
* 111.234 222.568
* **111.234 222.5678**
* An exception is thrown at runtime.

1. **What is the result?**



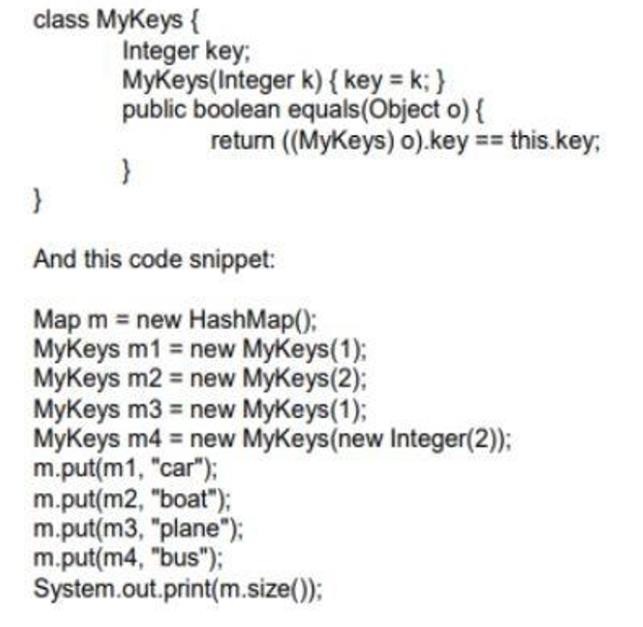
* Square square = new Square ("bar"); square.foo ("bar"); square.foo();
* Square square = new Square ("bar"); square.foo ("bar"); square.foo ("bar");
* Square square = new Square (); square.foo (); square.foo(bar);
* **Square square = new Square (); square.foo (); square.foo("bar");**
* Square square = new Square (); square.foo (); square.foo ();

1. **Which three implementations are valid?**



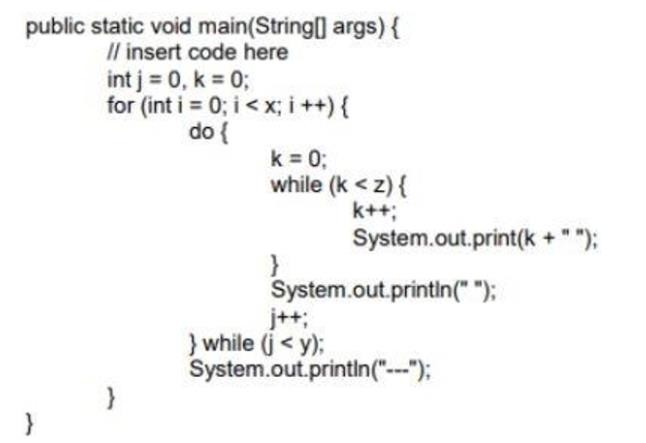
* **class Test implements SampleCloseable { public void close() throws java.io.IOException { // do something } }**
* class Test implements SampleCloseable { public void close() throws Exception { // do something } }
* **class Test implements SampleCloseable { public void close() throws FileNotFoundException {// do something } }**
* class Test extends SampleCloseable { public void close() throws java.io.IOException { // do something } }
* **class Test implements SampleCloseable { public void close() { // do something }}**

1. **What is the result?**

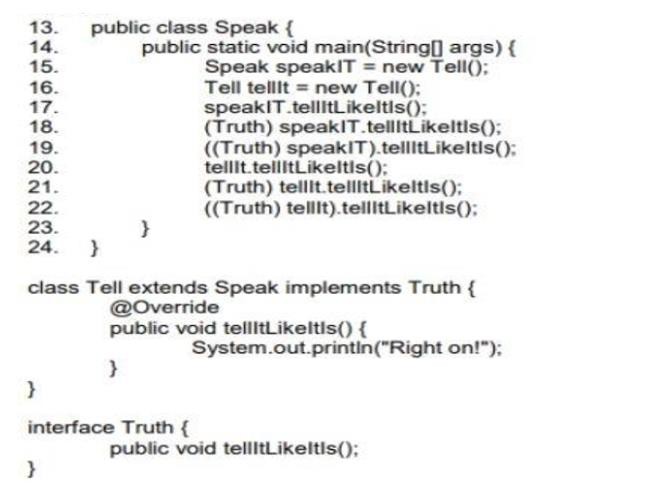


* 2
* 3
* **4**
* Compilation fails.

1. **What value of x, y, z will produce the following result? 1234,1234,1234 -----, 1234, ------**

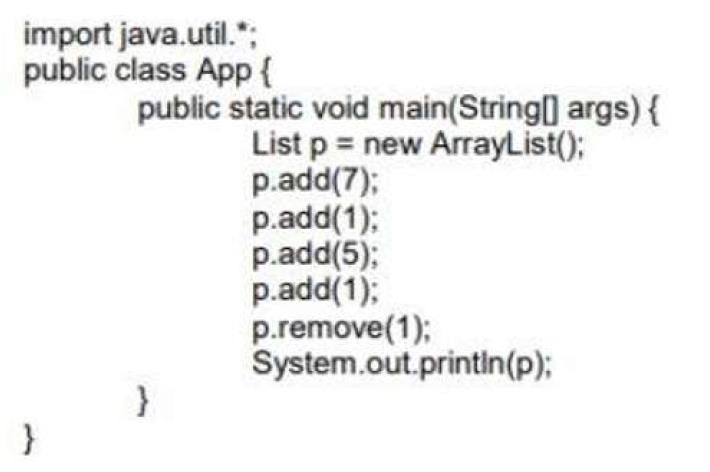


* int x = 4, y = 3, z = 2;
* int x = 3, y = 2, z = 3;
* int x = 2, y = 3, z = 3;
* **int x = 2, y = 3, z = 4;**
* int x = 4, y = 2, z= 3;

1. **Which three lines will compile and output "Right on!"?**

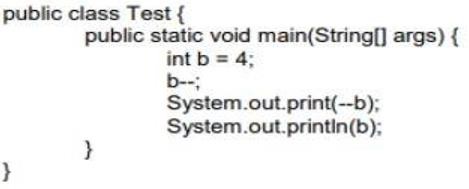
* Line 17
* Line 18
* **Line 19**
* **Line 20**
* Line 21
* **Line 22**

1. **¿Cuál es el resultado?**



* [7, 5]
* [7, 1]
* **[7, 5, 1]**
* [7, 1, 5, 1]

1. **What is the result?**

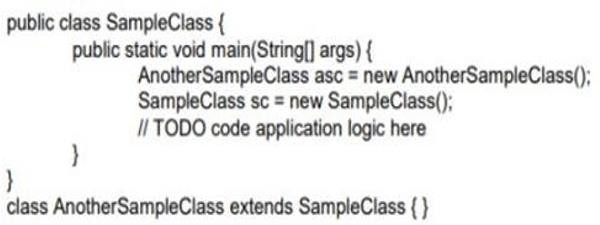


* **22**
* 12
* 32
* 33

1. **In Java the difference between throws and throw Is:**

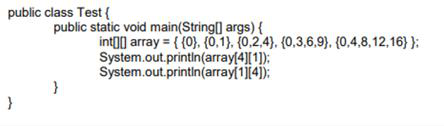
* Throws throws an exception and throw indicates the type of exception that the method.
* Throws is used in methods and throw in constructors.
* **Throws indicates the type of exception that the method does not handle and throw an exception.**

1. **Which statement, when inserted into line '' // TODO code application logic here", is valid in compilation time change?**



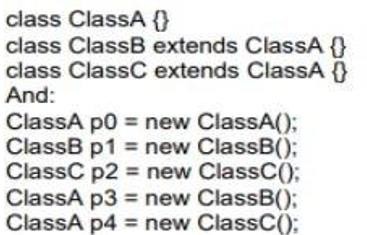
* asc = sc;
* **sc = asc;**
* asc = (Object) sc;
* asc= sc.clone();

1. **What is the result?**



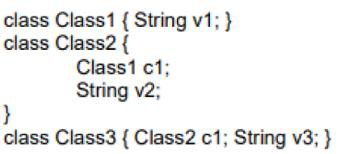
* 4 Null.
* Null 4.
* An IllegalArgumentException is thrown at run time.
* **4 An ArrayIndexOutOfBoundsException is thrown at run time**.

1. **Which three are valid? (Choose three)**



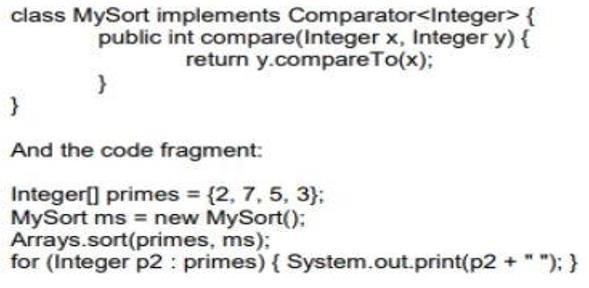
* **p0 = p1;**
* p1 = p2;
* p2 = p4;
* p2 = (ClassC)p1;
* **p1 = (ClassB)p3;**
* **p2 = (ClassC)p4;**

1. **Which three options correctly describe the relationship between the classes?**



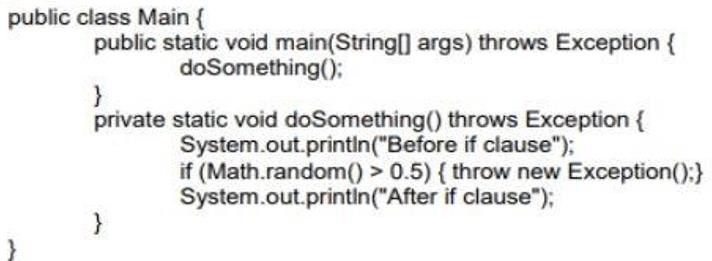
* Class2 has-a v3.
* Class1 has-a v2.
* **Class2 has-a v2.**
* **Class3 has-a v1.**
* Class2 has-a Class3
* **Class2 has-aClass1**

1. **What is the result?**



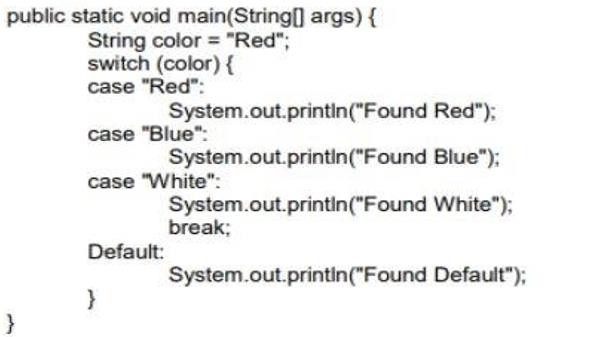
* 2 3 5 7
* 2 7 5 3
* **7 5 3 2**
* Compilation fails.

1. **Which two possible outputs?**



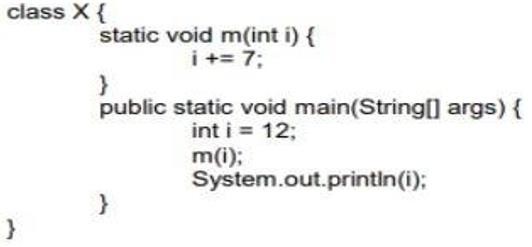
* Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).
* Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15) After if clause.
* **Exception in thread "main" java.lang.Exception at Main.doSomething (Main.java:21) at Main.main (Main.java:15).**
* **Before if clause After if clause.**

1. **What is the result?**



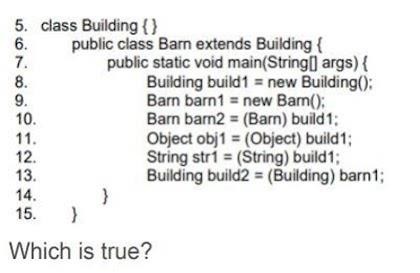
* Found Red.
* Found Red Found Blue.
* **Found Red Found Blue Found White**.
* Found Red Found Blue Found White Found Default.

1. **What is the result?**



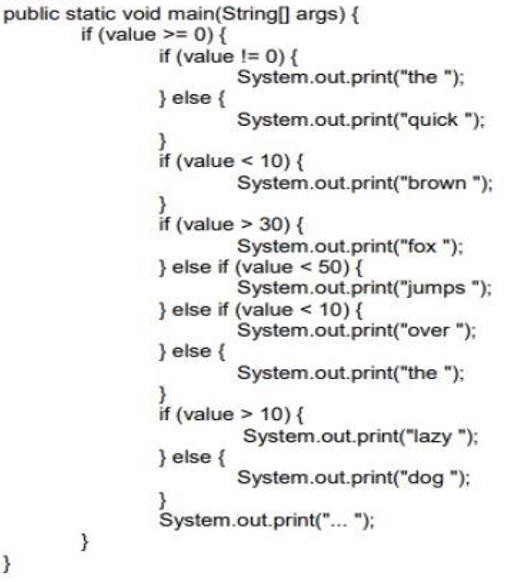
* 7
* **12**
* 19
* Compilation fails.
* An exception is thrown at run time

1. **Which is true?**



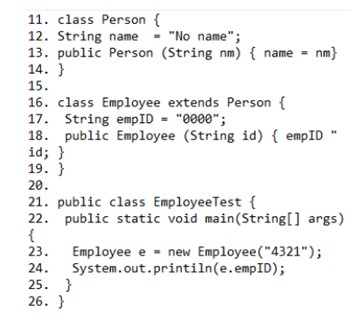
* If line 10 is removed, the compilation succeeds.
* If line 11 is removed, the compilation succeeds.
* **If line 12 is removed, the compilation succeeds.**
* If line 13 is removed, the compilation succeeds.
* More than one line must be removed for compilation to succeed.

1. **What is the result if the integer value is 33?**



* The fox jump lazy...
* **The fox lazy...**
* Quick fox over lazy...
* Quick fox the...

1. **What is the result?**

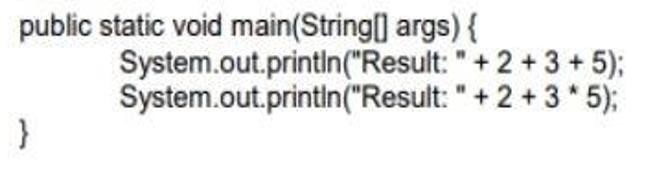


* 4321.
* 0000.
* An exception is thrown at runtime.
* **Compilation fails because of an error in line 18**.

1. **Which code fragment is illegal?**

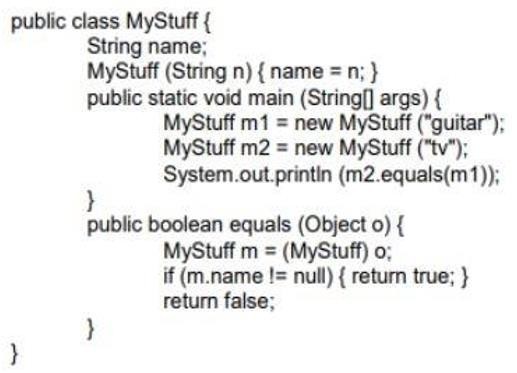
* Class Base1 { abstract class Abs1 { } }
* Abstract class Abs2 { void doit() { } }
* class Base2 { abstract class Abs3 extends Base2 { } }
* **class Base3 { abstract int var1 = 89; }**

1. **What is the result?**



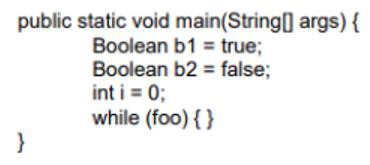
* Result: 10 Result: 30
* Result: 25 Result: 10
* **Result: 235 Result: 215**
* Result: 215 Result: 215
* Compilation fails.

1. **What is the result?**



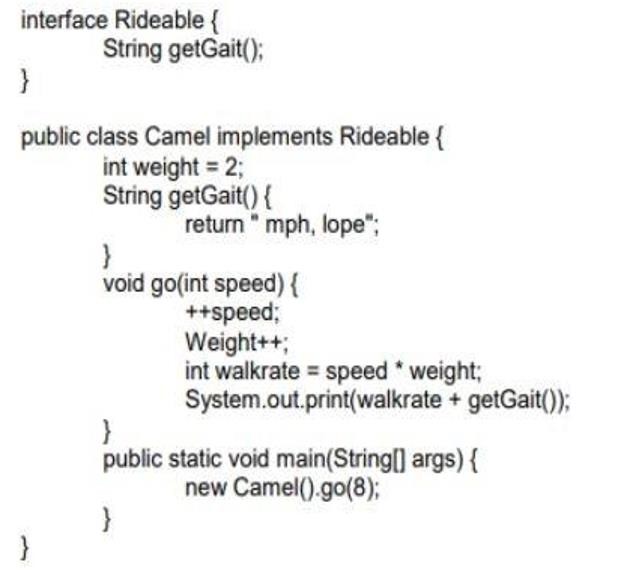
* The output is true and MyStuff fulfills the Object.equals() contract
* The output is false and MyStuff fulfills the Object.equals() contract
* **The output is true and MyStuff does NOT fulfill the Object.equals() contract.**
* The output is false and MyStuff does NOT fulfill the Object.equals() contract

1. **Which one is valid as a replacement for foo?**



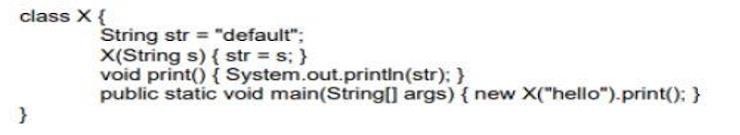
* b1.compareTo(b2)
* i = 1
* i == 2? -1:0
* **foo.equals("bar")**

1. **What is the result?**



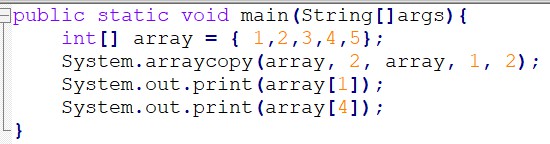
* 16 mph, lope
* 24 mph, lope.
* **Compilation fails**
* 27 mph, lope.

1. **What is the result?**



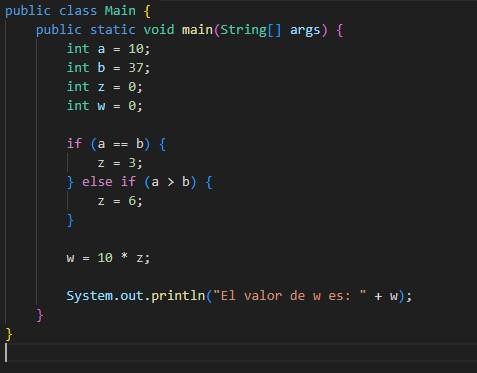
* **Hello**
* Default
* Compilation fails.
* The program prints nothing.
* An exception is thrown at run time.

1. **What is the result?**



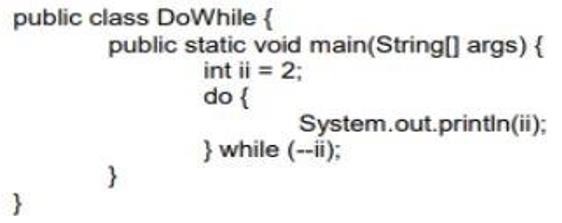
* 14
* 15
* 24
* 25
* 34
* **35**

1. **What is the result?**



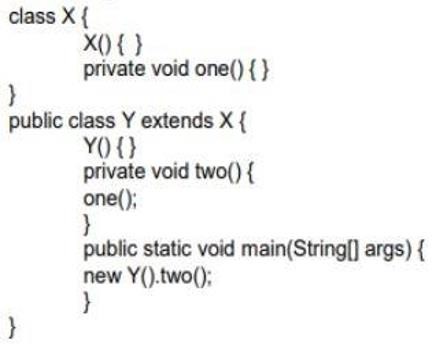
* **0**
* 30
* 60

1. **What is the result?**



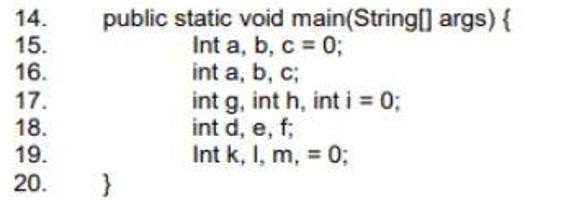
* 2 1 2
* 1 0
* null
* An infinite loop.
* **Compilation fails**

1. **What changes will make this code compile?**



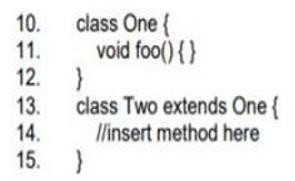
* Adding the public modifier to the declaration of class X.
* Removing the Y() constructor.
* Removing the private modifier from the two() method.
* Adding the protected modifier to the X() constructor.
* **Changing the private modifier on the declarationof the one() method to protected**.

1. **Which two declarations will compile?**



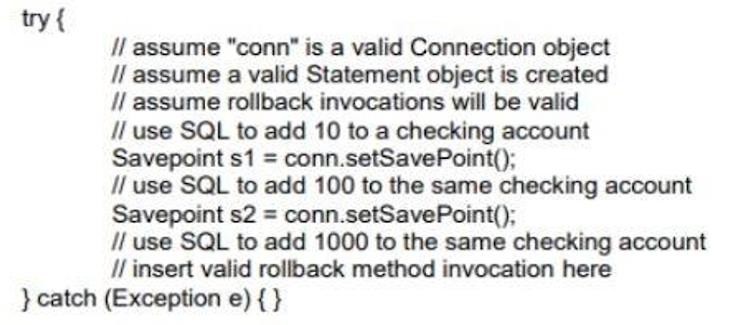
* Line 15.
* **Line 16**.
* Line 17.
* **Line 18**.
* Line 19
* Line 20

1. **Which three methods, inserted individually at line, will correctly complete class Two(Choose three)?**



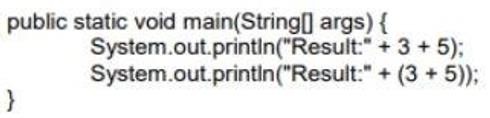
* **public void foo() { /\* more code here \*/ }**
* private void foo() { /\* more code here \*/ }
* **protected void foo() { /\* more code here \*/ }**
* **int foo() { /\* more code here \*/ }**

1. **What is the result?**



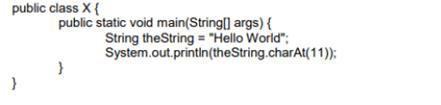
* **If conn.rollback(s1) is inserted, account will be incremented by 10**.
* If conn.rollback(s1) is inserted, account will be incremented by 1010.
* If conn.rollback(s2) is inserted, account will be incremented by 100
* **If conn.rollback(s2) is inserted, account will be incremented by 110**.
* If conn.rollback(s2) is inserted, account will be incremented by 1110

1. **Whats is the result?**



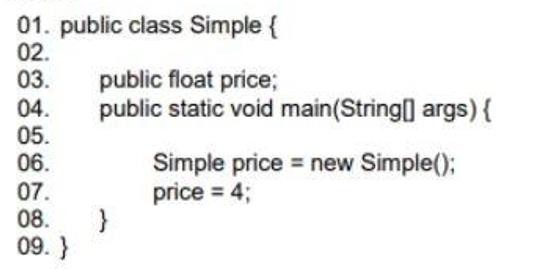
* Result: 8 Result: 8
* **Result: 35 Result: 8**
* Result: 8 Result: 35
* Result: 35 Result: 35

1. **What is the result?**



* There is no output.
* d is output.
* **A StringIndexOutOfBoundsException is thrown at runtime**.
* An ArrayIndexOutOfBoundsException is thrown at runtime.
* A NullPointException is thrown at runtime.
* A StringArrayIndexOutOfBoundsException is thrown at runtime.

1. **What will make this code compile and run?**

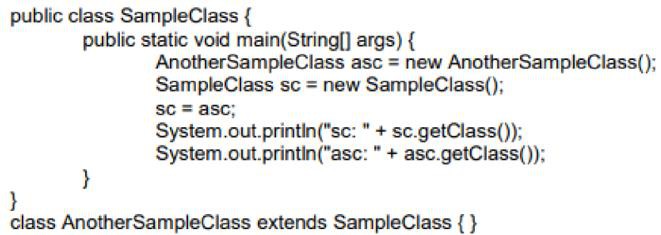


* **Change line 3 to the following: public int price**;
* Change line 7 to the following: int price = new Simple();
* Change line 7 to the following: float price = new Simple ();
* Change line 7 to the following: price = 4f;
* **Change line 7 to the following: price.price = 4**;

1. **In the Java collections framework a Set is:**

* **A collection that cannot contain duplicate elements**.
* An ordered collection that can contain duplicate elements
* An object that maps value key sets and cannot contain values Duplicates

1. **What is the result?**



* sc: class.Object asc: class.AnotherSampleClass
* sc: class.SampleClass asc: class.AnotherSampleClass
* sc: class.AnotherSampleClass asc: class.SampleClass
* **sc: class.AnotherSampleClass asc: class.AnotherSampleClass**

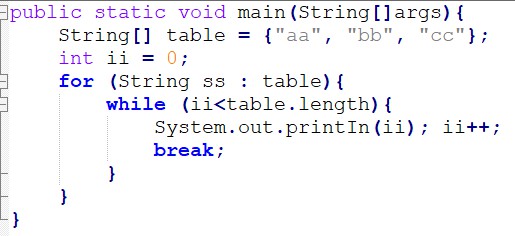
1. **Which declaration initializes a boolean variable?**

* **boolean j = (1 < 5);**
* boolean m = null;
* boolean h = 1;
* boolean k = 0;

1. **Which two will compile, and can be run successfully usaing the following command? Java Fred1 Hello walls.**

* abstract class Fred1 {public static void main(String[] args){System.out.println(args[2]);}}
* class Fred 1{ public static void main(String args) { System.out.println(args[1]); } }
* class Fred1 { public static void main(String[] args) { System.out.println(args[1]); }
* class Fred1 { public static void main(String[] args) { System.out.println(args); } }

1. **How many times is 2 printed?**



* Thrice
* It is not printed because compilation fails
* Twice
* Zero
* **Once**

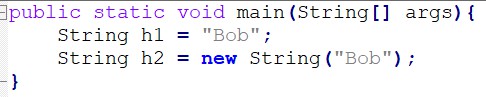
1. **Which two are valid instantiations and initializations of a multidimensional array?**

* **array2D[0][0] = 1; array2D[0][1] = 2; array2D[1][0] = 3; array2D[1][1] = 4**;
* array3D[0][0] = array; array3D[0][1] = array; array3D[1][0] = array; array3D[0][1] =array;
* int[][] array2D = {0, 1};
* **int[][][] array3D = {{0, 1}, {2, 3}, {4, 5}}; int[] array = {0, 1}; int[][][] array3D = newint[2][2][2**];
* int [][] array2D = {{0, 1, 2, 4}{5, 6}}; int[][] array2D = new int[2][2];

1. **The standard API for accessing databases in Java is:**

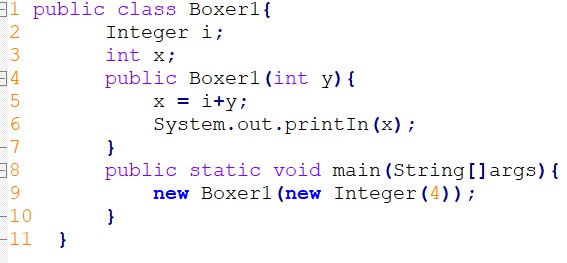
* JPA/Hibernate
* **JDBC**
* ODBC

1. **What is the best way to test that the values of h1 and h2 are the same?**



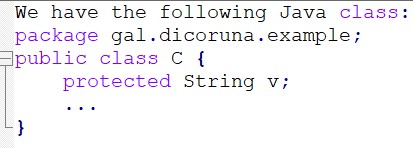
* **if (h1.equals(**h2)).
* if (h1 == h2).
* if (h1.toString() == h2.toString()).
* if (h1.same(h2)).

1. **What is the result?**



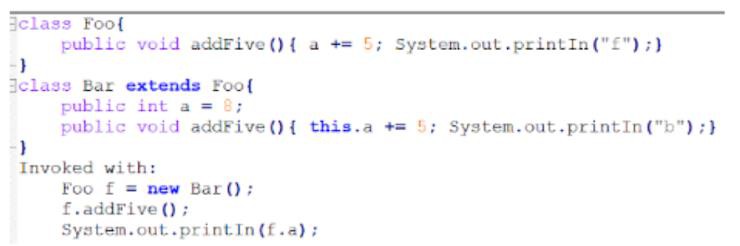
* **A NullPointerException occurs at runtime**.
* The value "4" is printed at the command line.
* An IllegalStateException occurs at runtime.
* Compilation fails because of an error in line 9.
* Compilation fails because of an error in line 5.
* A NumberFormatException occurs at runtime.

1. **Given:**



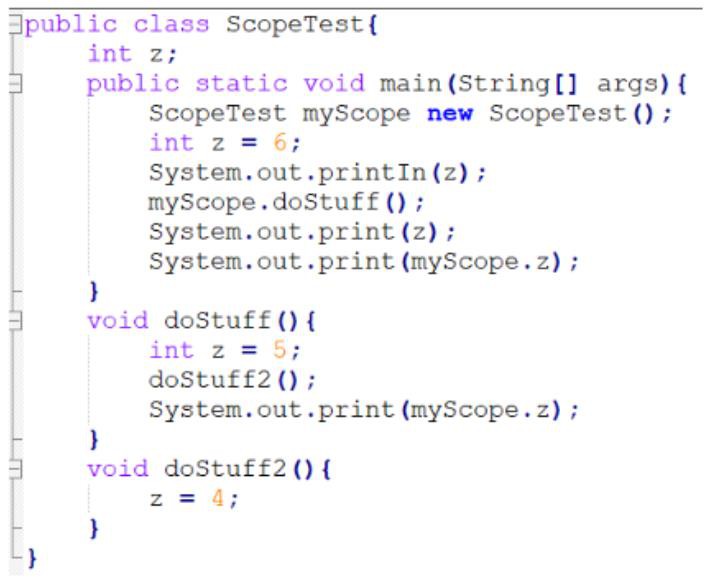
* **From the class, the package, subclasses**.
* Any site but only read if it is outside the package.
* From the class, the package, subclasses and all sites.

1. **What is the result?**



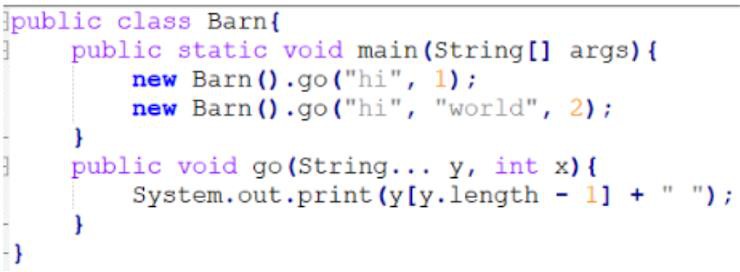
* b 3
* **Compilation fails**
* f 8
* b 8
* f 13
* f 3
* An exception is thrown at runtime. b 13

1. **What is the result?**



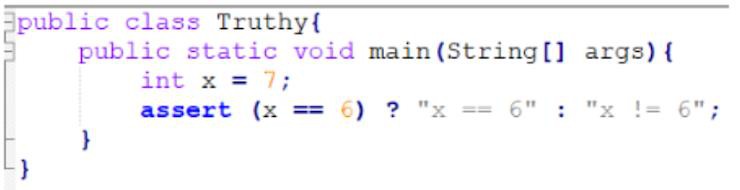
* 6554
* 6564
* 6565
* **Compilation fails**
* 6566

1. **What is the result?**



* worl world.
* hi hi
* hi world
* An exception is thrown at runtime.
* **Compilation fails**

1. **What is the result if you try to compile Truthy.java and then run it with assertionsenabled?**

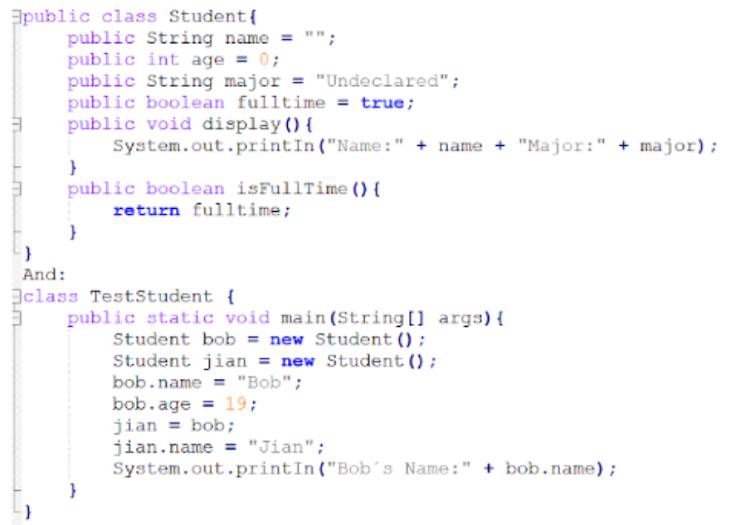


* Truthy.java compiles and the output is x != 6
* Truthy.java compiles and an AssertionError is thrown with x != 6 as additionaloutput.
* **Truthy.java does NOT compile.**
* Truthy.java compiles and an AssertionError is thrown with no additional output

1. **What is the result? Given:**

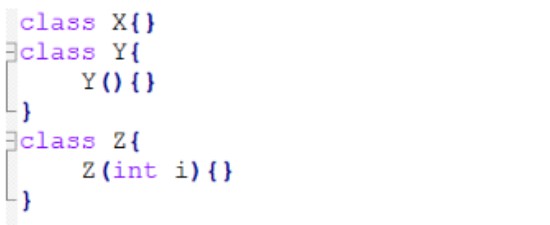
* 0 tom 0 jerry 1 tom 1 jerry 2 tom 2 jerry
* **0 tom 0 jerry 1 tom 2 tom 2 jerry**
* 0 tom 0 jerry 2 tom 2 jerry
* 0 tom 0 jerry 1 tom 1 jerry

1. **What is the result when this program is executed?**



* Nothing prints
* Bob's name.
* Bob's Name: Bob.
* **Bob's Name: Jian**

1. **Which class has a default constructor?**



* Z only.
* X only.
* X, Y and Z
* **X and Y**
* X and Z
* Y only
* Y and Z

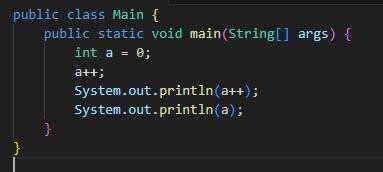
1. **Which two may precede the word "class" in a class declaration?**

* **Public**
* **Static**
* Synchronized
* Local
* Volatile

1. **The BUILDER pattern is used to:**

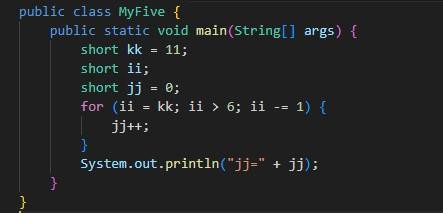
* Having several constructor methods in a class.
* **Simplify the creation of complex objects**.
* Implement the constructor of a class.

1. **What is the result?**



* 0 1
* **1 2**
* 2 2
* 1 1

1. **What value should replace kk in the comment to cause jj = 5 to be output?**

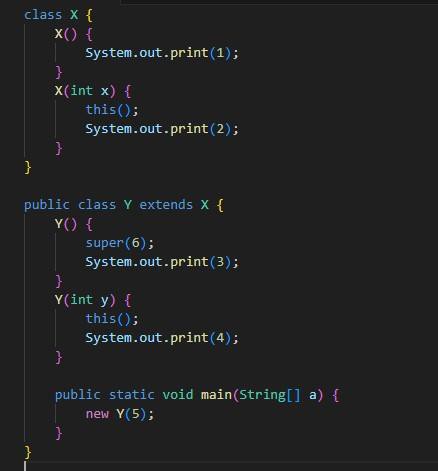


* -1
* 1
* 5
* 8
* **11**

1. **What is a static block of code in Java?**

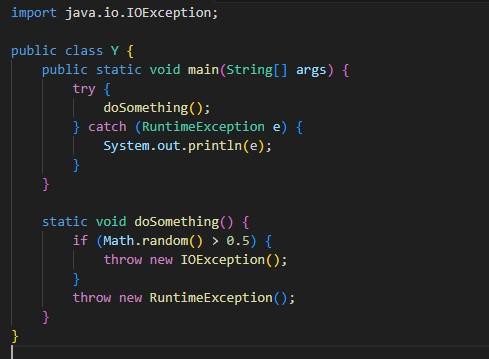
* A block of code within a class that runs whenever the class is load on the JVM.
* **A block of code inside a class that runs when that class first loaded in JVM**
* A block of code within a class that always runs before the builder.

1. **What is the result?**



* 2134
* 4321
* 2143
* 13
* **1213**
* 134

1. **Which two actions, used independently, will permit this class to compile?**

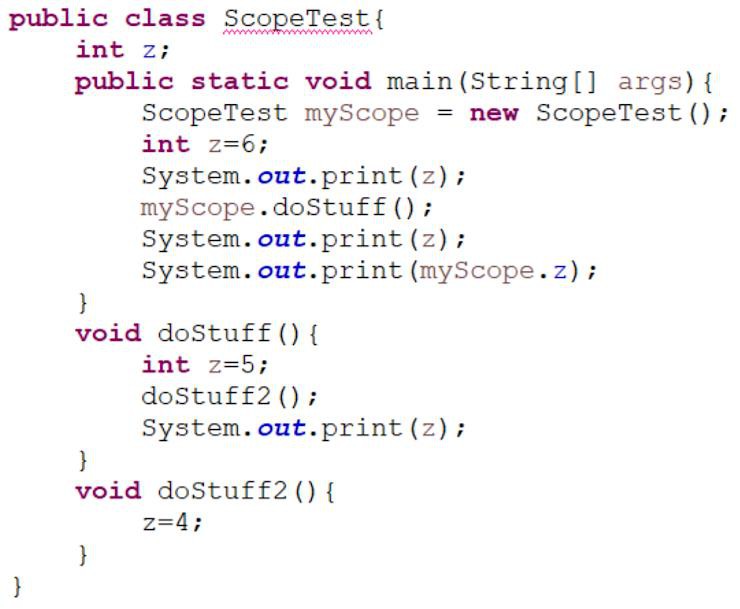


* **Adding throws IOException to the main() method ....IOException**
* Adding throws IOException to the dosomething() method signature and changing the catch
* **Adding throws IOException to the main() method signature and to the dosomething() method**
* Adding throws IOException to the main() method signature
* Adding throws IOException to the doSoomething() method signature.

1. **A method is declared to take three arguments. A program calls this method andpasses only two arguments.What is the result?**

* **Compilation fails**
* The third argument is given the value void.
* The third argument is given the value zero
* An exception occurs when the method attempts to access the third argument. The third argument is given the appropriate false value for its declared type. The third argument is given the value null.

1. **What is the result?**



* 6565
* 6504
* **6560**
* 6550

**Cuestionario 2**

1. **¿Qué es una API?**

         interfaz de programación de aplicaciones

1. **¿Qué es REST?**JavaScript Object Notation
2. **¿Que es un verbo HTTP?**HTTP define un conjunto de métodos o verbos de petición para indicar la acción que se desea realizar para un recurso determinado
3. **Menciona al menos 3 verbos en HTTP**GET, POST, PUT
4. **¿En términos REST que significa idempotencia?**La ejecución repetida de una petición con los mismos parámetros sobre un mismo recurso tendrá el mismo efecto en el estado de nuestro recurso en el sistema si se ejecuta 1 o N veces
5. **Menciona cuales son los verbos idempotentes**GET, PUT, DELETE
6. **Menciona que verbo no es idempotente.**POST
7. **Un verbo POST se utiliza para eliminar un recurso**

Falso

1. **Un verbo PATCH no se utiliza para actualizar un recurso de manera parcial.**

Falso

1. **Verbo HTTP que se utiliza para obtener/listar información**GET
2. **¿Qué es un código de retorno HTTP?**El código HTTP es en el que se indica si se ha completado satisfactoriamente o no una solicitud HTTP específica.
3. **¿Qué códigos de retorno HTTP se clasifican como informativos?**100
4. **¿Qué códigos de retorno HTTP se clasifican como de redireccionamiento?**300
5. **¿Qué códigos de retorno HTTP se clasifican como de respuesta satisfactoria?**200
6. **¿Qué códigos de retorno HTTP se clasifican como error de cliente?**400
7. **¿Qué códigos de retorno HTTP se clasifican como error del servidor?**500
8. **Un código de error 400 se utiliza cuando hay un error en el servidor**Verdadero

Falso

1. **Un código 201 se utiliza para devolver una respuesta satisfactoria pero con un contenido vacío.**Verdadero
2. **¿A qué le llamamos endpoint en un servicio REST?**Son las URLs de un API o un backend que responden a una petición.
3. **¿Cuál es la estructura correcta de un endpoint?**GET apiRest/v0/products}
4. **¿Qué es un Query Param en un servicio REST?**Son parámetros que son enviados al final del endpoint y siempre deberán estar después de un signo de interrogación.
5. **¿Qué es un UriParam en un servicio REST?**Parámetro de ruta, se usa básicamente para identificar un recurso o recursos específicos**.**

**ASO**

1. **Comando Git para mostrar el historial de commits**git log
2. **Comando GIT para crear un rama**git checkout -b
3. **Comando UNIX/Linux para borrar un archivo**rm
4. **Comando para iniciar un contenedor en Docker**docker start
5. **Comando de UNIX/Linux para mostrar la ruta actual**pwd
6. **Comando UNIX/Linux para crear una carpeta**mkdir
7. **Selecciona el comando Git correcto según la definición: "Crea un nuevo commit que deshace todos los cambios hechos en el commit, luego los aplica a la rama actual"**git revert
8. **Comando de UNIX/Linux para mover un archivo**mv
9. **Comando de UNIX/Linux para copiar un archivo**cp
10. **Comando para eliminar un contenedor en Docker**docker rm
11. **Comando Git para crear un repositorio**git init
12. **Comando para detener un contenedor en Docker**

docker stop

1. **Comando UNIX/Linux para listar el contenido de una carpeta**

Is

1. **Comando UNIX/Linux para moverte entre los directorios**

cd

1. **Comando Git para agregar archivos al Staging  
   git add**
2. **Comando para reescribir el historial de Git**git commit -amend

**HTTP—----------------------------------------------------------------------**

1. **Que significa el código 400:** «Bad Request»
2. **El verbo POST:**

Se utiliza para enviar una entidad a un recurso en específico, causando a menudo un cambio en el estado o efectos secundarios en el servidor.

1. **Que significa el código 200:** «OK»
2. **WSDL:** Web Services Description Language
3. **Que significa el código 204:** «No Content»
4. **El verbo PUT:** reemplaza todas las representaciones actuales del recurso de destino con la carga útil de la petición.
5. **El verbo PATCH:** Es utilizado para aplicar modificaciones parciales a un recurso.
6. **El verbo GET:** solicita una representación de un recurso específico
7. **Que significa el código 500:** «Internal Server Error»
8. **Que significa el código 404:** «Not Found»
9. **Que es idempotencia:** Es la propiedad de realizar una acción varias veces y conseguir siempre el mismo resultado
10. **DELETE:** borra un recurso en específico.
11. **WADL:** Web Application Description Language.
12. **Cual de los siguientes verbos HTTP no es idempotente:** POST
13. **504 :** «Gateway Timeout»

**Conceptos y Spring\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

1. **Nombra los métodos de inyección de dependencias  
   -** Inyección vía constructor

-Inyección vía métodos

 -Inyección vía propiedades

1. **Maven permite:**

**-**Creación de documentación

-Creación de informes

-Resolución de dependencias

-Releases Distribuciones

-Compilaciones

-Todas las anteriores

1. **POM:** Project Object Model, es un fichero XML, que es la unidad principal de un proyecto Maven
2. **Selecciona  los puntos correctos sobre: API (Selecciona 3):  
   -** significa Application Programming Interface  
   - permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados  
   - simplifica el desarrollo de aplicaciones
3. **Cual de las siguientes afirmaciones es correcta sobre la inyección de dependencias:**Es un patrón de desarrollo de software donde los objetos no son responsables de inicializar sus dependencias
4. **Nombra los 3 estereotipos básicos de Spring**@Repository  
   @Service  
   @Controller
5. **Selecciona  los puntos correctos sobre: JSON (Selecciona 3)**es un formato ligero de intercambio de datos  
   significa JavaScript Object Notation  
   para las máquinas es simple interpretarlo y generarlo
6. **Que descripción se adapta mejor a la definición del Contenedor de Spring?**Se encarga de crear los objetos, conectarlos entre sí, configurarlos y además controla los ciclos de vida de cada uno objeto mediante el patrón de inyección de dependencias
7. **Notación para quitar los warnings**@SuppressWarnings
8. **Nombra los 3 scopes básicos de un Bean**Singleton  
   Singleton  
   Request

**42. Selecciona la definición de scope SESSION**El contenedor de Spring creará una nueva instancia del objeto definido por el Bean para cada una de las sesiones HTTP (Aplicación Stateful) y entregará esa misma instancia cada vez que reciba una petición dentro de la misma sesión.

**Información adicional**

**Funciones Principales de Postman**

Envío de Solicitudes HTTP:

* GET: Solicitar datos de un servidor.
* POST: Enviar datos para crear un nuevo recurso.
* PUT: Actualizar un recurso existente.
* DELETE: Eliminar un recurso.

El verbo HTTP **PATCH** se utiliza precisamente para actualizar un recurso de manera parcial. A diferencia de **PUT**, que generalmente se utiliza para actualizar un recurso completo, **PATCH** permite modificar sólo una parte del recurso. Esto es útil cuando solo necesitas cambiar ciertos campos de un recurso sin tener que enviar toda la representación del recurso.

**Diferencias Entre PATCH y PUT**

* **PATCH**: Se usa para aplicar cambios parciales a un recurso. Sólo necesitas enviar los datos que deseas modificar.
  + Ventajas: Menor carga de datos, especialmente útil para grandes recursos.
  + Ejemplo: Cambiar solo el campo email de un usuario.
* **PUT**: Se usa para actualizar completamente un recurso. Generalmente, envías toda la representación del recurso, incluso los campos que no se modifican.
  + Ventajas: Facilita la sustitución completa del recurso.
  + Ejemplo: Actualizar todo el objeto usuario.

### Categorías de Códigos de Estado HTTP

Los códigos de estado HTTP están organizados en cinco categorías, cada una representada por el primer dígito del código de tres cifras:

1. **1xx (Informativos)**:
   * Indican que la solicitud ha sido recibida y el proceso está en curso.
   * **100 Continue**: El servidor ha recibido los encabezados de la solicitud y el cliente debe continuar enviando el cuerpo de la solicitud (si lo hay).
2. **2xx (Éxito)**:
   * Indican que la solicitud se ha recibido, entendido y procesado con éxito.
   * **200 OK**: La solicitud ha sido exitosa.
   * **201 Created**: La solicitud ha sido exitosa y ha resultado en la creación de un nuevo recurso.
   * **204 No Content**: La solicitud ha sido exitosa, pero no hay contenido que devolver.
3. **3xx (Redirección)**:
   * Indican que se necesita tomar una acción adicional por parte del cliente para completar la solicitud.
   * **301 Moved Permanently**: El recurso solicitado ha sido movido permanentemente a una nueva URL.
   * **302 Found**: El recurso solicitado está temporalmente en una ubicación diferente, pero el cliente debe usar la URL original para futuras solicitudes.
   * **304 Not Modified**: El recurso no ha sido modificado desde la última solicitud.
4. **4xx (Errores del Cliente)**:
   * Indican que la solicitud contiene errores que impiden que el servidor la procese.
   * **400 Bad Request**: La solicitud está mal formada o es ilegible.
   * **401 Unauthorized**: Se requiere autenticación para acceder al recurso.
   * **403 Forbidden**: El servidor entiende la solicitud, pero se niega a autorizarla.
   * **404 Not Found**: El recurso solicitado no se ha encontrado en el servidor.
5. **5xx (Errores del Servidor)**:
   * Indican que el servidor ha encontrado una condición que le impide completar la solicitud.
   * **500 Internal Server Error**: Ha ocurrido un error genérico en el servidor.
   * **502 Bad Gateway**: El servidor, actuando como puerta de enlace o proxy, ha recibido una respuesta no válida del servidor de respaldo.
   * **503 Service Unavailable**: El servidor no está disponible temporalmente, generalmente debido a mantenimiento o sobrecarga.

**¿Que es idempotencia?**En un sentido general, se refiere a una propiedad que posee una operación o función, la cual, cuando se aplica múltiples veces al mismo elemento o conjunto de elementos, produce el mismo resultado que si se aplicara una sola vez.  
  
En el contexto de la informática y las API (Interfaces de Programación de Aplicaciones), la idempotencia es una propiedad deseable para ciertas operaciones. Cuando una operación es idempotente, realizarla una vez o varias veces tiene el mismo efecto; es decir, el resultado final es el mismo.  
La idempotencia es una propiedad que asegura que aplicar una operación una o múltiples veces no altera el resultado final más allá de la primera aplicación de dicha operación. Esto es crucial en sistemas distribuidos y en el diseño de APIs para garantizar la integridad y consistencia de los datos.

**Definición del Contenedor de Spring**El contenedor de Spring, también conocido como contenedor IoC (Inversión de Control) o contenedor de aplicaciones, es uno de los aspectos fundamentales del marco de trabajo Spring. Proporciona un entorno de ejecución para las aplicaciones de Spring, gestionando la creación, configuración y administración de los objetos (beans) de la aplicación.  
Aquí tienes una definición detallada del contenedor de Spring:

1. **Gestión de Objetos (Beans)**: El contenedor de Spring es responsable de crear y gestionar los objetos de la aplicación, conocidos como beans. Estos beans pueden ser objetos de negocio, componentes de acceso a datos, servicios web, o cualquier otro componente de la aplicación.
2. **Inyección de Dependencias**: El contenedor de Spring aplica el patrón de Inyección de Dependencias (DI), lo que significa que se encarga de proveer las dependencias requeridas por un bean cuando éste es creado. Esto promueve la modularidad y la desacoplamiento entre los componentes de la aplicación.
3. **Configuración Declarativa**: Spring permite configurar los beans de la aplicación de forma declarativa, utilizando XML, anotaciones o código Java. Esta configuración se utiliza para definir los beans, sus dependencias y cómo se deben inicializar.
4. **Gestión del Ciclo de Vida de los Beans**: El contenedor de Spring controla el ciclo de vida de los beans, inicializándolos cuando son necesarios y liberando los recursos asociados cuando ya no son necesarios. Esto garantiza una gestión eficiente de los recursos y la eliminación de posibles problemas de manejo de memoria.
5. **Aspectos y Transacciones**: El contenedor de Spring también puede administrar aspectos, que son aspectos de la funcionalidad de la aplicación que se aplican de manera transparente, como la gestión de transacciones. Esto simplifica la implementación de funcionalidades comunes y mejora la modularidad de la aplicación.

En resumen, el contenedor de Spring es el núcleo del marco de trabajo Spring, proporcionando un entorno de ejecución flexible y poderoso para la construcción de aplicaciones empresariales robustas y escalables. Su capacidad para gestionar objetos, inyectar dependencias, configurar declarativamente y administrar el ciclo de vida de los beans es fundamental para el desarrollo efectivo de aplicaciones con Spring.

**Métodos de inyección de dependencias**Más comunes en Spring son:

1. \*\*Inyección por Constructor\*\*: En este método, las dependencias requeridas por una clase se pasan al constructor de la clase. Spring utiliza el constructor de la clase para crear una instancia y proporcionar las dependencias necesarias.

2. \*\*Inyección por Setter\*\*: En este método, las dependencias requeridas se proporcionan a través de métodos de "setter" en la clase. Spring llama a estos métodos de setter después de crear una instancia de la clase y antes de devolverla como un bean.

3. \*\*Inyección por Campo (Field Injection)\*\*: En este método, las dependencias requeridas se anotan directamente en los campos de la clase. Spring utiliza reflexión para asignar valores a estos campos después de crear una instancia de la clase.

Cada uno de estos métodos tiene sus propias ventajas y desventajas, y su elección depende de las necesidades específicas del proyecto y las preferencias de diseño del desarrollador. Sin embargo, el método preferido en Spring es la inyección por constructor debido a que promueve una mejor inmutabilidad y hace que las dependencias requeridas sean explícitas en la creación del objeto.