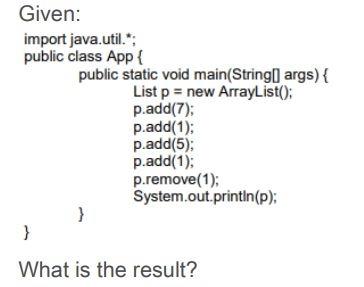
**Examen Java-APX Estudio**

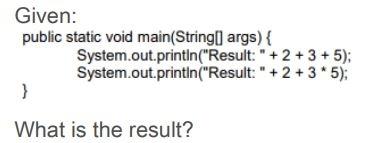
**Guía de Estudio para Examen Java**

**Q1)**



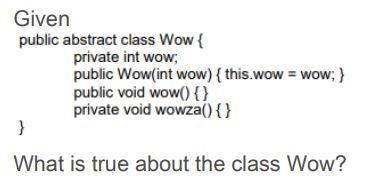
1. [7, 1, 5, 1]
2. [7, 5, 1]
3. [7, 5]
4. [ 7, 1]

**Q2)**



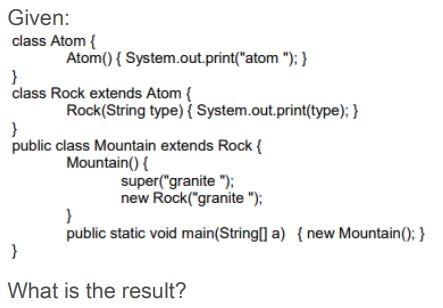
1. Result: 10 Result: 30
2. Result: 25 Result: 10
3. Result: 235 Result: 215
4. Result: 215 Result: 215
5. Compilation fails.

**Q3)**



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

**Q4)**



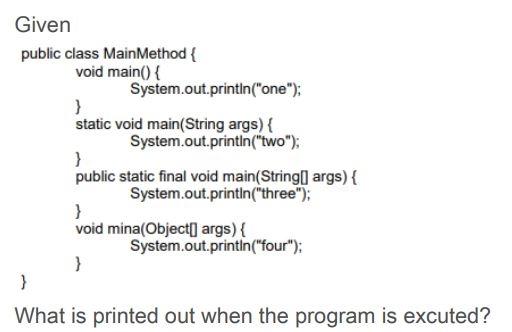
1. Compilation fails.
2. Atom granite.
3. Granite granite.
4. Atom granite granite.
5. An exception is thrown at runtime.
6. Atom granite atom granite.

**Q5)**

What is the DTO pattern used for?

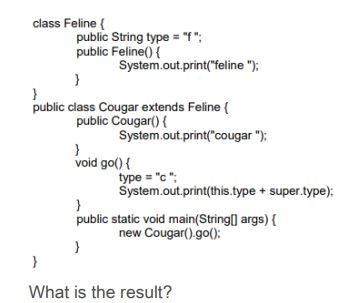
1. To implement the data access layer
2. To exchange data between processes
3. To implement the presentation layer.

**Q6)**



1. one
2. two
3. three
4. four
5. There is no output.

**Q7)**



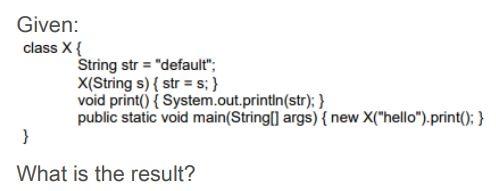
1. Cougar c f.
2. Feline cougar c c.
3. Feline cougar c f.
4. Compilation fails.

**Q8)**

The pom.xml file is the configuration file For:

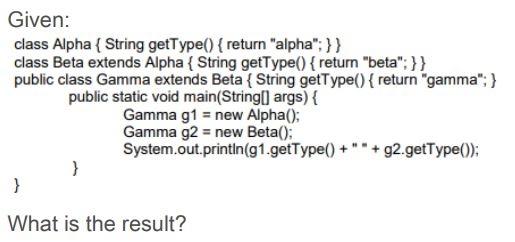
1. Ant
2. Gradle
3. Maven

**Q9)**



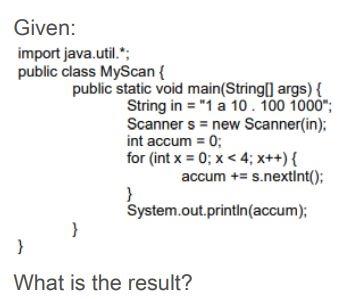
1. Hello
2. Default
3. Compilation fails.
4. The program prints nothing.
5. An exception is thrown at run time.

**Q10)**



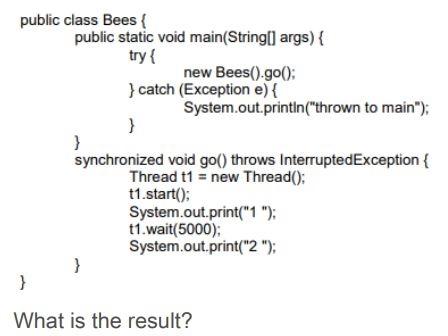
* Alpha beta
* Beta beta.
* Gamma gamma.
* Compilation fails.

**Q11)**



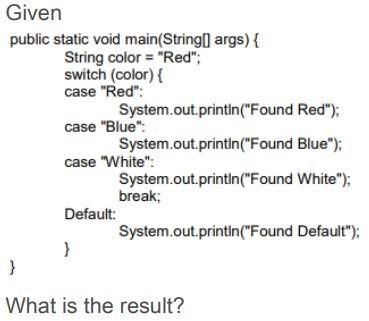
* 11
* 111
* 1111
* An exception is thrown at runtime.

**Q12)**



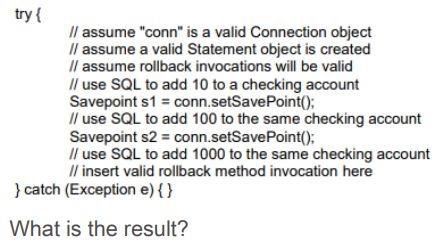
* The program prints 1 then 2 after 5 seconds.
* The program prints: 1 thrown to main.
* The program prints: 1 2 thrown to main.
* The program prints:1 then t1 waits for its notification.

**Q13)**



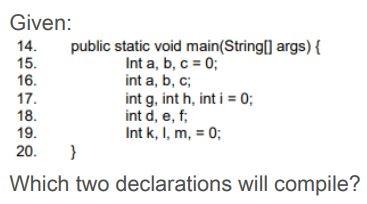
* Found Red.
* Found Red Found Blue.
* Found Red Found Blue Found White.
* Found Red Found Blue Found White Found Default.

**Q14)**



* 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

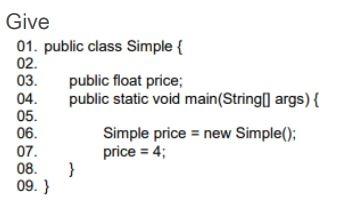
Q15)



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

**Q16)**

What will make this code compile and run?



Change line 3 to the following: public int price;

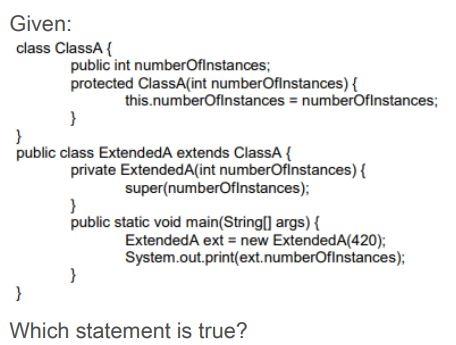
Change line 7 to the following: int price = new Simple();

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;

**Q17)**



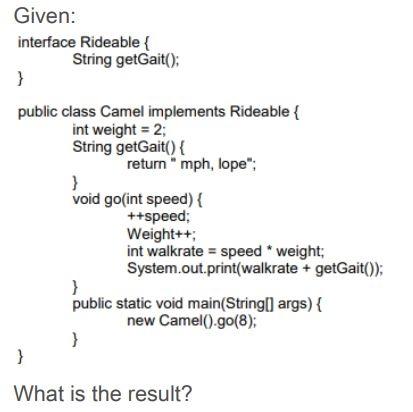
* 420 is the output.
* An exception is thrown at runtime.
* All constructors must be declared public.
* Constructors CANNOT use the private modifier.
* Constructors CANNOT use the protected modifier.

**Q18)**

The SINGLETON pattern allows:

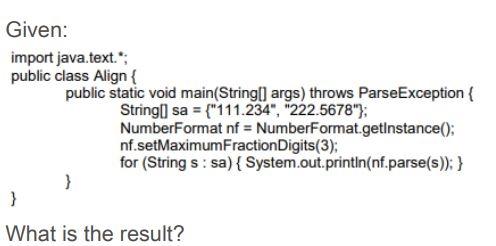
* Have a single instance of a class and this instance cannot be used by other classes
* Having a single instance of a class, while allowing all classes have access to that instance.
* Having a single instance of a class that can only be accessed by the first method that calls it.

**Q19)**



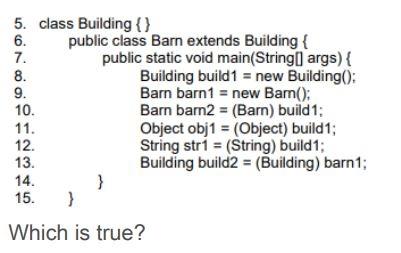
* 16 mph, lope
* 24 mph, lope.
* 27 mph, lope.
* Compilation fails

**Q20)**



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

**Q21)**

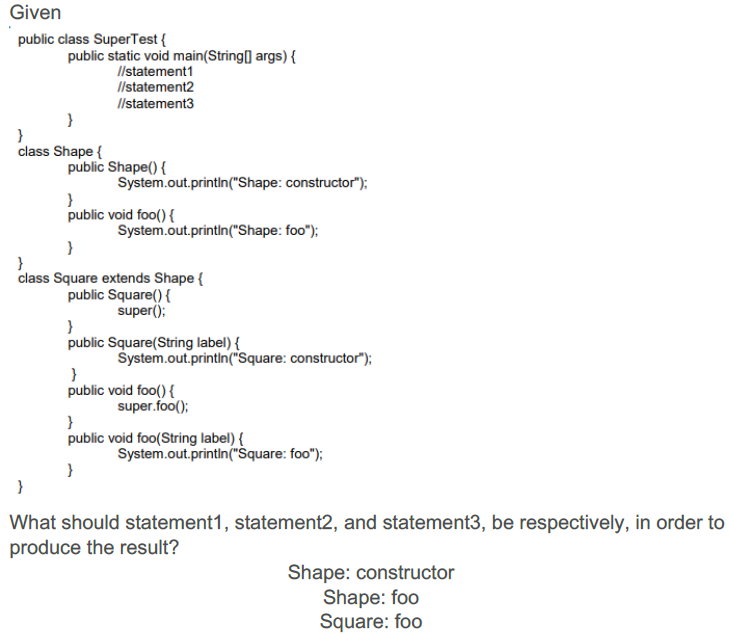


* If line 10 is removed, the compilation succeeds.
* If line 11 is removed, the compilation succeeds.
* If line 12 is removed, the compilation succeeds.
* If line 13 is removed, the compilation succeeds.
* More than one line must be removed for compilation to succeed.

**Q22)**

Which two statements are true?

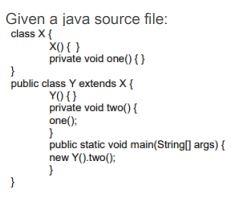
* An abstract class can implement an interface.
* An abstract class can be extended by an interface.
* An interface CANNOT be extended by another interface.
* An interface can be extended by an abstract class.
* An abstract class can be extended by a concrete class.
* An abstract class CANNOT be extended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

**Q23)** 

* Square square = new Square ("bar"); [square.foo](http://square.foo/) ("bar"); [square.foo](http://square.foo/)();
* Square square = new Square ("bar"); [square.foo](http://square.foo/) ("bar"); [square.foo](http://square.foo/) ("bar");
* Square square = new Square (); [square.foo](http://square.foo/) (); [square.foo](http://square.foo/)(bar);
* Square square = new Square (); [square.foo](http://square.foo/) (); [square.foo](http://square.foo/)("bar");
* Square square = new Square (); [square.foo](http://square.foo/) (); [square.foo](http://square.foo/) ();

**Q24)**

What changes will make this code compile?



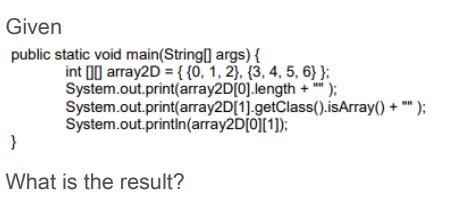
* Adding the public modifier to the declaration of class X. (
* Adding the protected modifier to the X() constructor.
* Changing the private modifier on the declarationof the one() method to protected.
* Removing the Y() constructor
* Removing the private modifier from the two() method.

**Q25)**

Which code fragment is illegal?

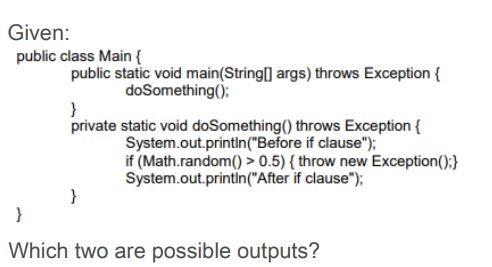
* Class Base1 { abstract class Abs1 { } }
* Abstract class Abs2 { void doit() { } }
* class Base2 { abstract class Abs3 extends Base2 { } }
* class Base3 { abstract int var1 = 89; }

**Q26)**



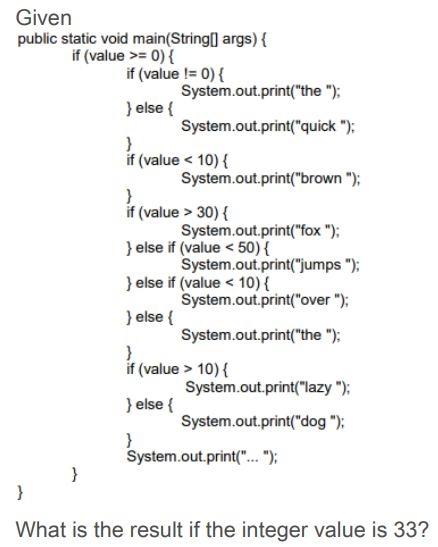
* 3false1
* 2true3
* 2false3
* 3true1
* 3false3
* 2true1
* 2false1

**Q27)**



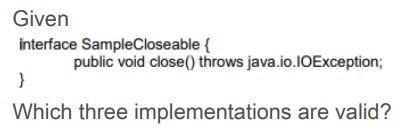
* Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething ([Main.java:21](http://main.java:21/)) at Main.main ([Main.java:15](http://main.java:15/)).
* Before if clause Exception in thread "main" java.lang.Exception at Main.doSomething ([Main.java:21](http://main.java:21/))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

**Q28)**



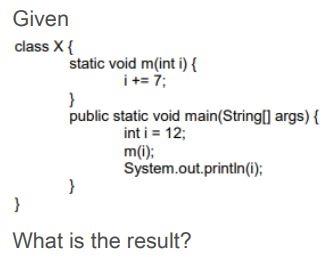
* The fox jump lazy ?
* The fox lazy ?
* Quick fox over lazy ?

**Q29)**



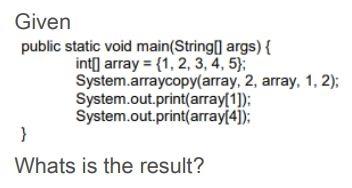
* class Test implements SampleCloseable { public void close() throws [java.io](http://java.io/).IOException { // do something } }
* class Test implements SampleCloseable { public void close() throws Exception { // do something } }
* class Test implements SampleCloseable { public void close() throws FileNotFoundException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

**Q30)**



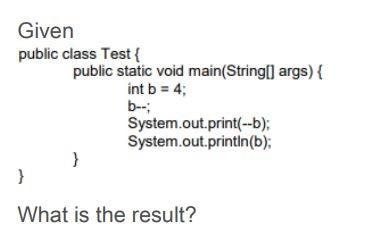
* 7
* 12
* 19
* Compilation fails.
* An exception is thrown at run time

**Q31)**



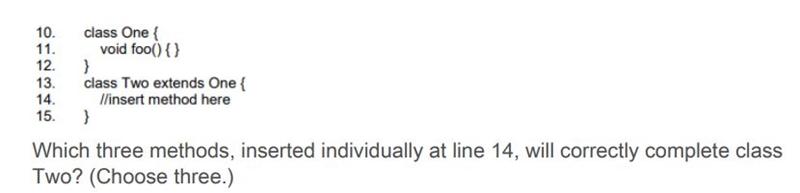
* 14
* 15
* 24
* 25
* 34
* 35

**Q32)**



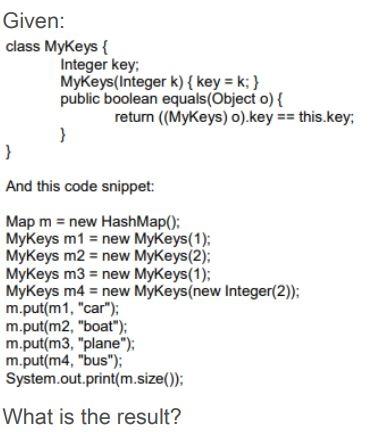
* 22
* 12
* 32
* 33

**Q33)**



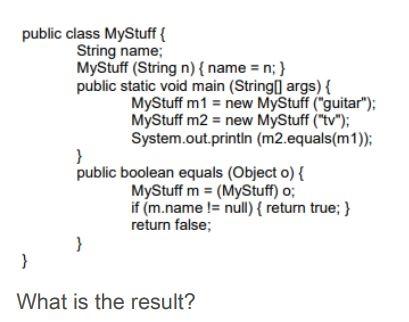
* int foo() { /\* more code here \*/ }
* void foo() { /\* more code here \*/ }
* public void foo() { /\* more code here \*/ }
* private void foo() { /\* more code here \*/ }
* protected void foo() { /\* more code here \*/ }

**Q34)**



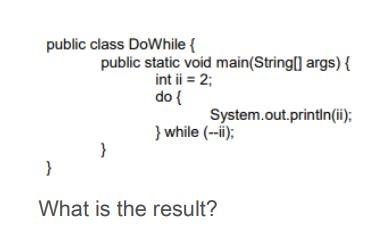
* 2
* 3
* 4
* Compilation fails.

**Q35)**



* 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

**Q36)**



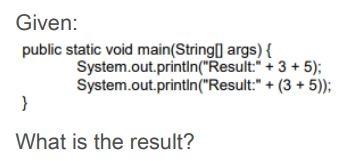
* 2 1 2
* 1 0
* null
* An infinite loop.
* Compilation fails

**Q37)**

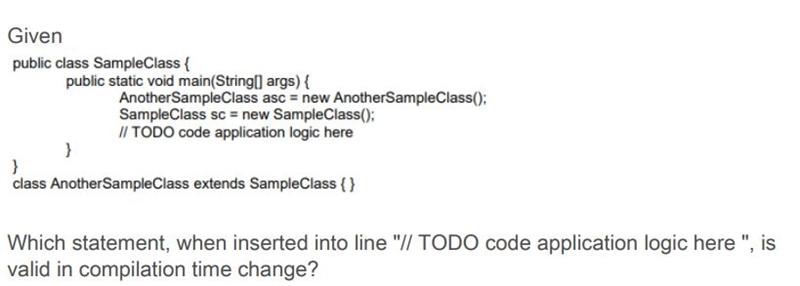
In the Java collections framework a Set Is:

* 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

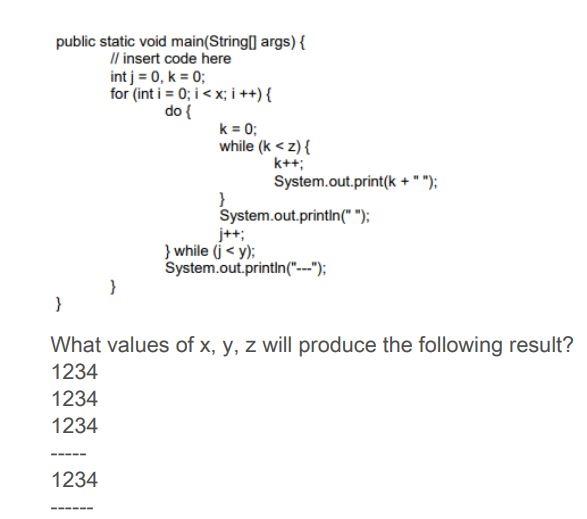
**Q38)**



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

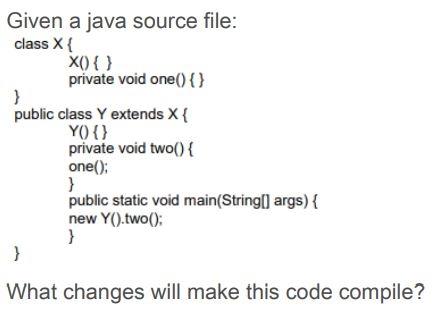
**Q39)**

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

**Q40)**

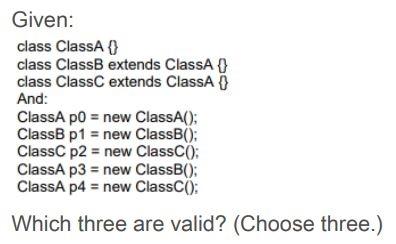
* int x = 4, y = 3, z = 2;
* int x = 3, y = 2, z = 3;
* int x = 2, y = 3, z = 3;
* int x = 4, y = 2, z= 3;
* int x = 2, y = 3, z = 4;

Q41)



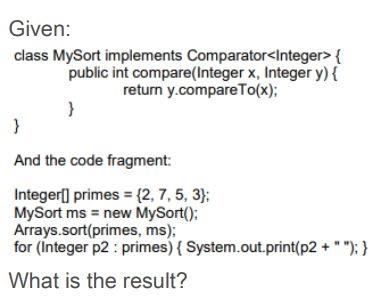
* Adding the public modifier to the declaration of class X.
* Adding the protected modifier to the X() constructor.
* Changing the private modifier on the declaration of the one() method to protected.
* Removing the Y() constructor.

**Q42)**



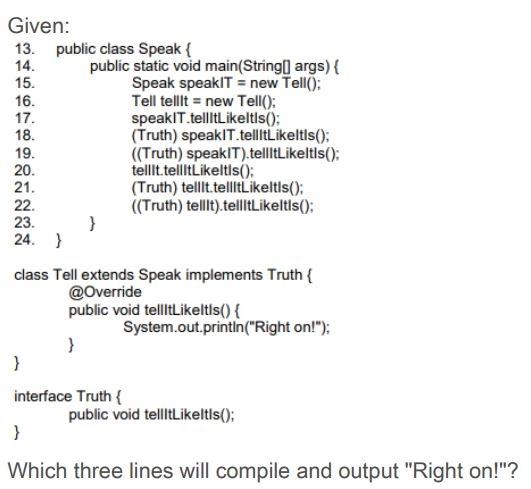
* p0 = p1;
* p1 = p2;
* p2 = p4;
* p2 = (ClassC)p1;
* p1 = (ClassB)p3;
* p2 = (ClassC)p4;

**Q43)**



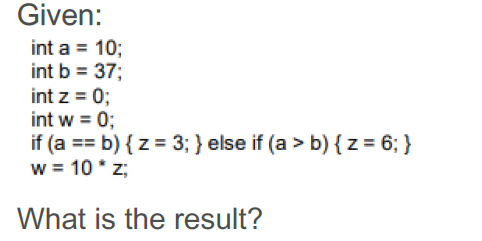
* 2 3 5 7
* 2 7 5 3
* 7 5 3 2
* Compilation fails.

**Q44)**



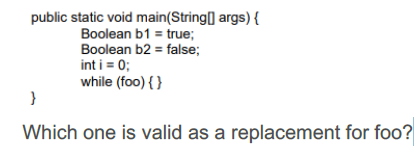
* Line 17
* Line18
* Line 19
* Line 20
* Line 21
* Line 22

**Q45)**



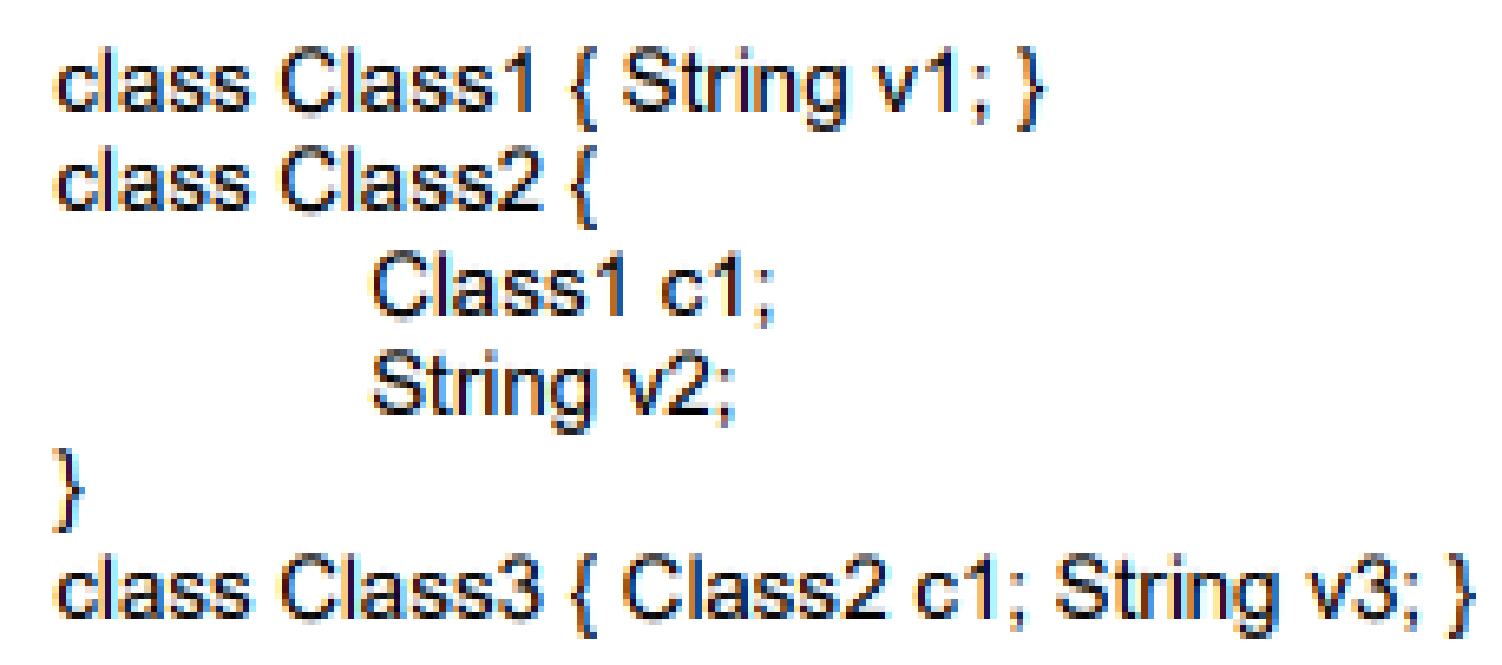
* 0
* 30
* 60

**Q46)**



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

**Q47)**

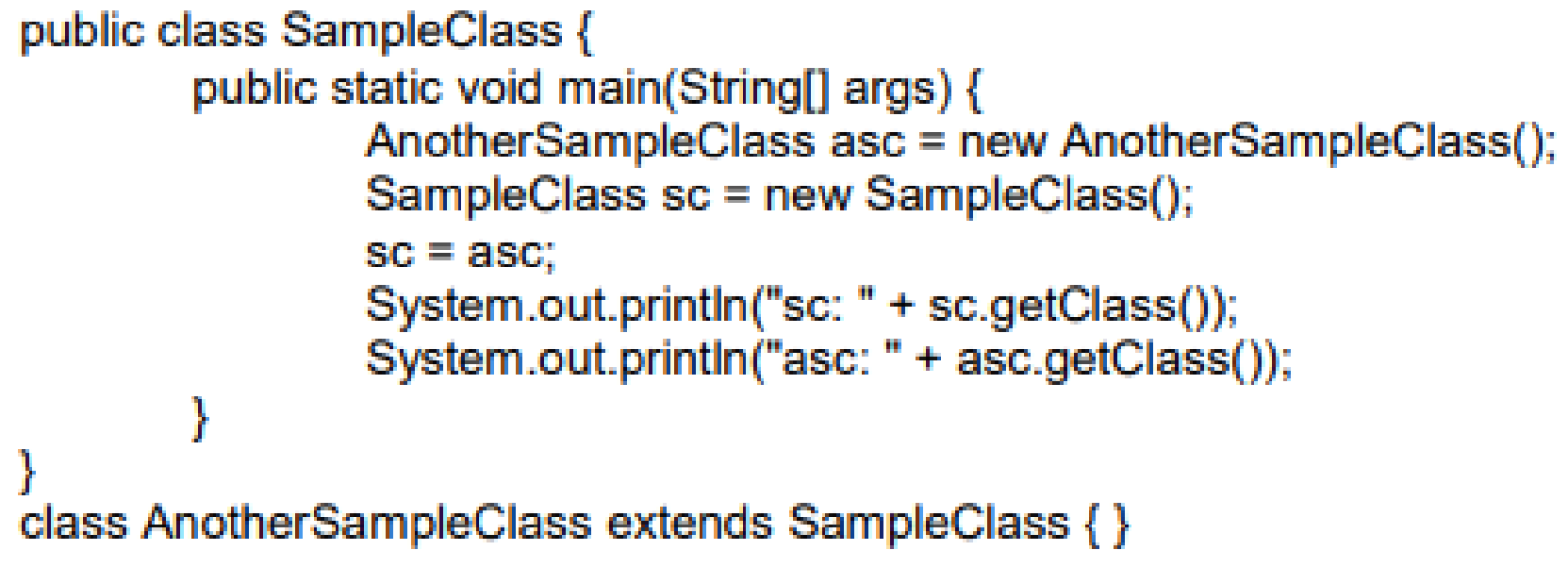


**Which three options correctly describe the relationship between the classes?**

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

**Q48)**

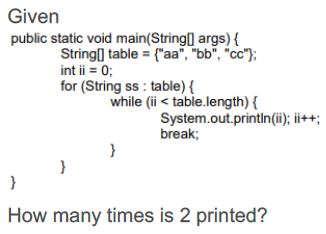
Given

****

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

**Q49)**



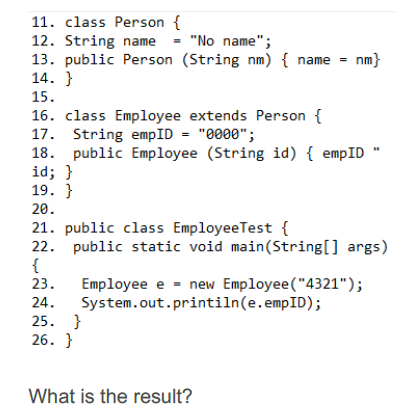
* Zero.
* Once.
* Twice.
* Thrice.
* It is not printed because compilation fails.

**Q50)**

In Java the difference between throws and throw Is:

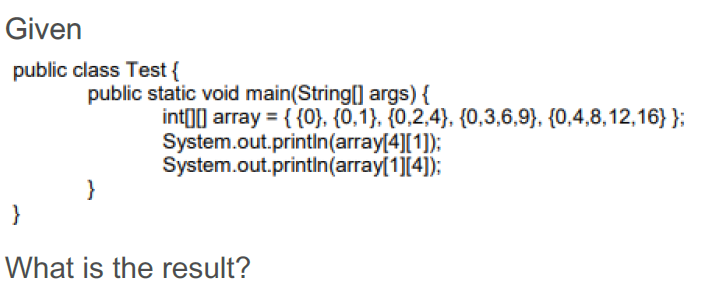
* Throws throws an exception and throw indicates the type of exception that the method.
* Throws is used in methods and throw in constructors.
* Throws indicates the type of exception that the method does not handle and throw an exception.

Q51)



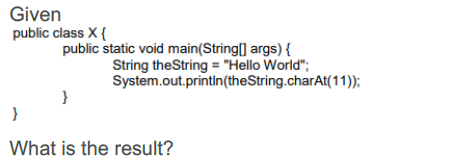
* 4321.
* 0000.
* An exception is thrown at runtime.
* Compilation fails because of an error in line 18.

**Q52)**



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

**Q53)**



* 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 runtim.