08 Java instanceof

Problema 1

Given the following class definitions and declaration:

```
class A {}
class B extends A {}
class C extends B {}
class D extends C {}

D d = new D();
```

the expression `(d instanceof A)` will return true.

You had to select 1 option

- True
- False

Explanation:
D extends C, which extends B, which extends A.
This means, D is-a C, C is-a B, and B is-a A. Therefore, D is-a A.
Hence, d instanceof A will return true.

Problema 2

If `a.equals(b)` returns `true`, `b instanceof ClassOfA` must always be true.

(Assume that `ClassOfA` is the name of the class of the variable `a`.)

You had to select 1 option

- True
- False

Explanation:

This may not always be correct because `equals()` method can be overridden. By default, it tests reference assignment, but any subclass of Object is free to redefine `equals()` as it deems fit. So, it is possible that an equals method may return `true` even if the class of the passed object has no relation to this object.

Problema 3

Which of the following statements are true?
You had to select 1 option

- For any non-null reference o1, the expression (o1 instanceof o1) will always yield true.
  - instanceof operator does not accept objects as the right hand side operand. The operand at the right hand side should be a class. Therefore, this expression will not compile.
- For any non-null reference o1, the expression (o1 instanceof Object) will always yield true.
  - Because all objects in Java derive from Object class.
- For any non-null reference o1, the expression (o1 instanceof o1) will always yield false.
  - It is wrong for the same reason as option 1.
- For any non-null reference o1, the expression (o1 instanceof Object) may yield false.
  - Since all objects in Java derive from Object class, there is no way it will ever return false.
- None of the above.

Problema 4

Given the following class definitions, the expression

```
(obj instanceof A) && ! (obj instanceof C) && ! (obj instanceof D)
```

correctly identifies whether the object referred to by obj was created by instantiating class B rather than classes A, C and D?

```
class A {}
class B extends A {}
class C extends B {}
class D extends C {}
```

You had to select 1 option

- True
- False

Explanation:

The given expression will not be able to distinguish between an object of class A and an object of class B. It will return true in both the cases. Also, The last part `!(obj instanceof D)` of the given expression is redundant because anything which is not instance of C cannot be an instanceof D either!

Correct expression would be `(obj instanceof B) && ! (obj instanceof C)`. This will return true only if obj points to an object of class B and not of A, C, or D.

Problema 5

Consider the following code:

```
interface Flyer{ }
class Bird implements Flyer { }
class Eagle extends Bird { }
class Bat { }

public class TestClass {

    public static void main(String[] args) {
        Flyer f = new Eagle();
        Eagle e = new Eagle();
        Bat b = new Bat();

        if(f instanceof Bird) System.out.println("f is a Bird");   // ✔
        if(e instanceof Flyer) System.out.println("e is a Flyer"); // ✔
        if(b instanceof Flyer) System.out.println("b is a Flyer"); //Compila
                                                                   // No error en Runtime
    }
}
```

What will be printed when the above code is compiled and run?

You had to select 1 option

- It will not compile.
- It will throw an exception when run.
- `f is a Bird`
  `e is a Flyer`
  - At run time, `'f'` points to an object of class `'Eagle '`. Now, `Eagle` extends `Bird` so `'f instanceof Bird'` returns `'true'`. `'e'` points to an object of class `'Eagle'`. Eagle extends Bird, which in turn implements Flyer. So an `'Eagle'` is a `'Flyer'`.
  - Therefore, `'e instanceof Flyer'` will also return `'true'`.
  - `'b'` points to an object of class Bat, which does not extend from Bird. Therefore, `'b instanceof Flyer'` returns `'false'`.
  - Dato curioso es que aunque no sea del tipo de la interfaz no hay error ni de compilación, ni de ejecución y simplemente regresa un `'false'` cuando si fuera una clase normal este marca error en la compilación.
- `f is a Bird`
- `e is a Flyer`

Problema 5 v2

Consider the following code:

```
interface Flyer{ }
class Bird implements Flyer { }
class Eagle extends Bird { }
class Bat { }

public class TestClass {

    public static void main(String[] args) {
        Flyer f = new Eagle();
        Eagle e = new Eagle();
        Bat b = new Bat();

        if(f instanceof Flyer) System.out.println("f is a Flyer"); // ✔
        if(e instanceof Bird) System.out.println("e is a Bird");   // ✔
        if(b instanceof Bird) System.out.println("b is a Bird");   // No compila
    }
}
```

What will be printed when the above code is compiled and run?

You had to select 1 option

- It will not compile.
    - b points to an object of class Bat, which does not extend from Bird. Now, it is possible for b to point to an object of any subclass of Bat. However, it is not possible for that sub class to extend Bird (because a class can at most extend from only one class).
    - Therefore, it is not possible for b to point to an object of a class that "is a" Bird. The compiler figures out this fact at compile time itself and so the code fails to compile.
- It will throw an exception when run.
- `f is a Flyer` `// ✔`
  `e is a Bird` `// ✔`
    - At run time, f points to an object of class Eagle. Now, Eagle extends Bird, which implements Flyer so f instanceof Flyer returns true.
    - e points to an object of class Eagle. Eagle extends Bird. Therefore, e instanceof Bird will also return true.
- `f is a Flyer`
- `e is a Bird`

Problema 6

Which of the given lines can be inserted at //1 of the following program ?

```
public class TestClass{
   public static void main(String[] args){
     short s = 9;
     //1
   }
}
```

You had to select 2 options

- `Short k = new Short(9); System.out.println(k instanceof Short);`
  - 9 is considered an int and there is no constructor in Short that takes an int. Short s = new Short((short) 9); will work.
- `System.out.println(s instanceof Short);`
  - The left operand of instanceof MUST be a reference variable and not a primitive.
- `Short k = 9; System.out.println( k instanceof s);`
  - Right operand of instanceof MUST be a 'reference type name', i.e., a class, an interface, or an enum name.
- `int i = 9; System.out.println(s == i);`
  - Any two numeric primitives can be compared using == operator. Even a numeric primitive and a primitive wrapper (for example, an int with a Short or a Double) can be compared. However, two wrappers of different types (for example, an Integer and a Short) cannot be compared using ==.
- `Boolean b = s instanceof Number;`
  - Left operand of instanceof MUST be an object and not a primitive.
- `Short k = 9; Integer i = 9; System.out.println(k == i);`
  - This will not compile because k and i are referring to objects that have no IS-A relationship among themselves.
- `Integer i = 9; System.out.println( s == i );`

Problema 7

Which of the given statements are correct about the following code?

```
//Filename: TestClass.java
class TestClass{
   public static void main(String[] args){
      A a = new A();
      B b = new B();
   };
}
class A implements T1, T2{}
class B extends A implements T1{} // T1 es innecesario, lo hereda de A
interface T1 { }
interface T2 { }
```

You had to select 4 options

- (a instanceof T1) will return true.
- (a instanceof T2) will return true.
- (b instanceof T1) will return true.
- (b instanceof T2) will return true.
- (b instanceof A) will return false.
    - It will return true because B extends A and 'b' is referring to an object of class B.

Explanation:

Since A implements both T1 and T2, 1 and 2 are correct.
b instanceof A will return true as B is a subclass of A. Note that it is 'A' and not 'a'.
`( b instanceof a )` will not compile.

Problema 8

Expression (s instanceof java.util.Date) will return false if 's' is declared as a variable of class java.lang.String.

You had to select 1 option

- True
- False

Explanation:

It will not even compile because the compiler knows that 's' (which is declared as of class `String`) can NEVER refer to an object of class `java.util.Date`. So, it will not accept this code.
Had 's' been declared as a variable of type `Object`, this code would have compiled because compiler sees that at run time it is possible for s to refer to an object of class `Date`.

Problema 9

Which letters will be printed when the following program is run?

```
public class TestClass{
  public static void main(String args[]){
    B b = new C();
    A a = b;
    if (a instanceof A) System.out.println("A");
    if (a instanceof B) System.out.println("B");
    if (a instanceof C) System.out.println("C");
    if (a instanceof D) System.out.println("D");
  }
}
class A { }
class B extends A { }
class C extends B { }
class D extends C { }
```

You had to select 3 options

- A will be printed.
- B will be printed.
- C will be printed.
- D will be printed.

Explanation:
// Hacen escaleras hasta C en tiempo de ejecución pro apuntar a `C()`
The program will print A, B and C when run. The object denoted by reference a is of type C. The object is also an instance of A and B, since C is a subclass of B and B is a subclass of A. The object is not an instance of D.

Problema 10

Consider :

```
class A {}
class B extends A {}
class C extends B {}
```

Which of these boolean expressions correctly identifies when an object 'o' actually refers to an object of class B and not of C?

You had to select 2 options

- `(o instanceof B) && (!(o instanceof A))`
  - This will return false if o refers to an Object of class A, B, or C because (o instanceof A) will be true for all the three.
- `!((o instanceof A) || (o instanceof B))`
- `(o instanceof B) && (!(o instanceof C))`
  - // Aquí dice que sea `'B'`, 'y' que NO sea `'C'`
- `! ( !(o instanceof B) || (o instanceof C))`
  - This is the complement of "(o instanceof B) && (!(o instanceof C))" prefixed with a '!'. So in effect, both are same.
  - // Aquí dice NO
    - NO es 'B' o es 'C'
  - // Puede verse como SI es `'B'` y NO es `'C'`, como dice arriba, es un complementario
- `(o instanceof B) && !((o instanceof A) || (o instanceof C))`

Explanation:
The expression `(o instanceof B)` will return `true` if the object referred to by `o` is of type B or a subtype of B.
The expression `(! (o instanceof C))` will return `true` unless the object referred to by `o` is of type C or a subtype of C.

Thus, the expression `(o instanceof B) && (!(o instanceof C))` will only return `true` if the object is of type B or a subtype of B that is not C or a subtype of C.
Given objects of classes A, B and C, this expression will only return `true` for objects of class B.