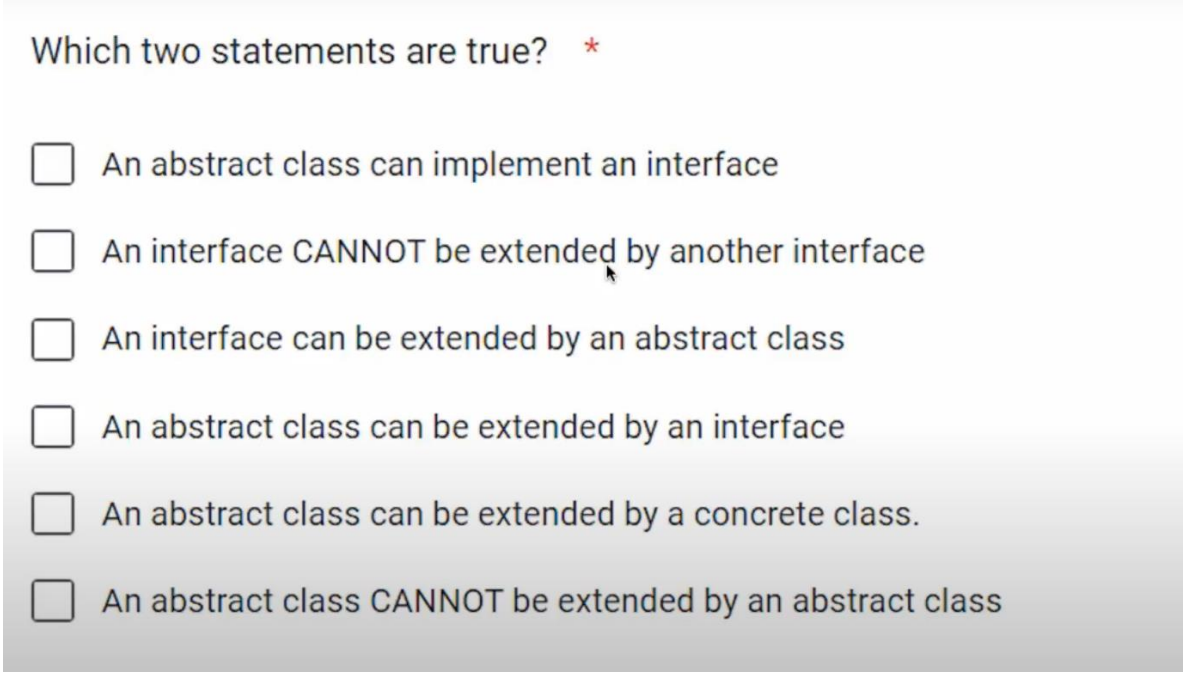
Natalia Denise Fuentes Rubio
Semana 3 – Evidencias
Academia Java

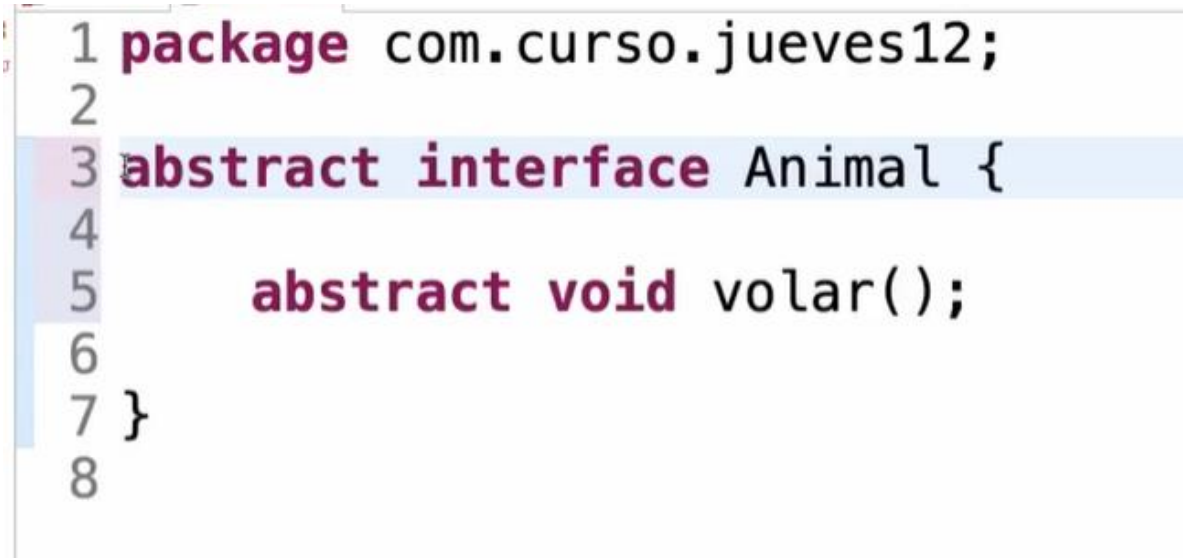
Problema Accenture, Which two sttements are true? Sintaxis en interfaces y clases abstractas.



- An abstract class can be extended by a concrete class.
- An abstract class can implement an interface

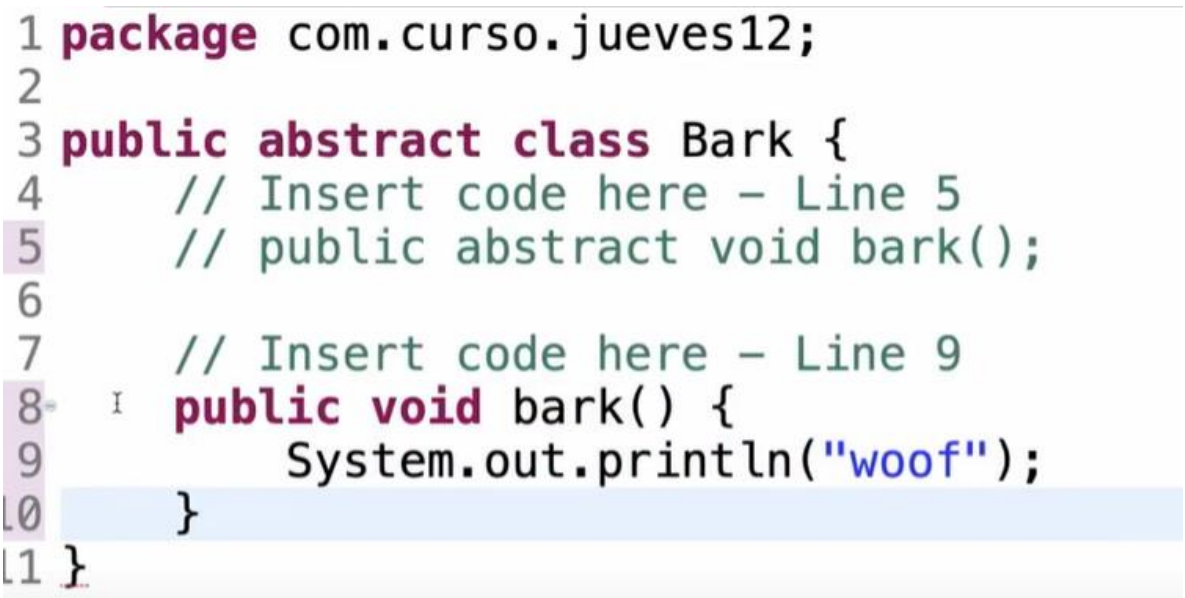
Método abstracto sólo puede ser definido en clase abstracta. // Sintaxis

Las interfaces siempre son abstractas, al igual que sus métodos, aunque no se definan como tal.



```
1 package com.curso.jueves12;
2
3 abstract interface Animal {
4
5     abstract void volar();
6
7 }
8
```

Las clases, por su parte, siempre se tienen que definir como *abstract*, pero sus métodos pueden o no ser definidos como abstractos, esto quiere decir que los métodos pueden tener comportamientos definidos.



```
1 package com.curso.jueves12;
2
3 public abstract class Bark {
4     // Insert code here - Line 5
5     // public abstract void bark();
6
7     // Insert code here - Line 9
8     public void bark() {
9         System.out.println("woof");
10    }
11 }
```

Problema "Bark" El problema requiere insertar código para que compile

```
1  package com.curso.jueves12;
2
3  public abstract class Bark {
4      // Insert code here - Line 5
5      public abstract void bark();
6
7      // Insert code here - Line 9
8      public void bark() {
9          System.out.println("woof");
10     }
11 }
```

Para el que código funcione, se debe definir una *class abtract Dog* que tenga el método *public abstract void bark();* y en la línea 9, se va a heredar la clase *Dog* para que pueda realizar el *bark* mediante un *Override*

```
1  package com.curso.jueves12;
2
3  abstract class Dog {
4      public abstract void bark();
5  }
6
7  class Poodle extends Dog{
8      @Override
9      public void bark() {
10         System.out.println("woof");
11     }
12 }
```

Bloques anónimos de instancia y static

- **Bloque anónimo:** Es un bloque que no tiene nombre, permite encadenar una o más sentencias que se ejecutan en secuencia. Se ejecuta primero que el constructor y va a la par de ejecución que estos.

```java
package com.curso.jueves12;

public class Principal {

    Principal() {
        System.out.println("Constructor Principal");
    }

    {
        System.out.println("Bloque anonimo");
    }

    public static void main(String[] args) {
        new Principal();
        new Principal();
        new Principal();
    }
```

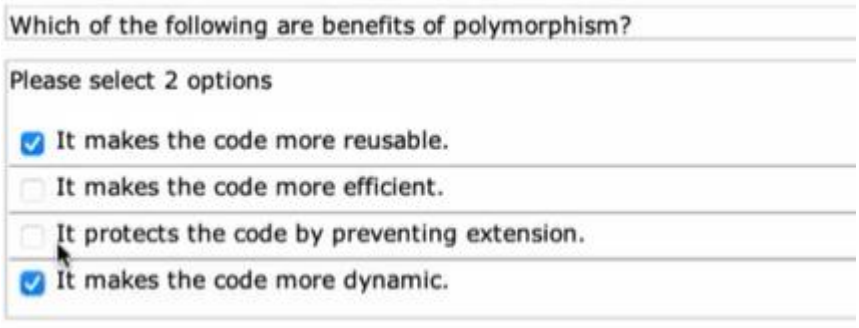
```
Bloque anonimo
Constructor Principal
Bloque anonimo
Constructor Principal
Bloque anonimo
Constructor Principal
```

Bloque anónimo static: Es el mismo bloque anónimo, pero al ser static, ya no le pertenece al constructor, sino que es parte de la clase como tal y solo se va a ejecutar una vez, antes que los constructores.

```java
package com.curso.jueves12;

public class Principal {

    Principal() {
        System.out.println("Constructor Principal");
    }

    static {
        System.out.println("Bloque anonimo");
    }

    public static void main(String[] args) {
        new Principal();
        new Principal();
        new Principal();
    }
```

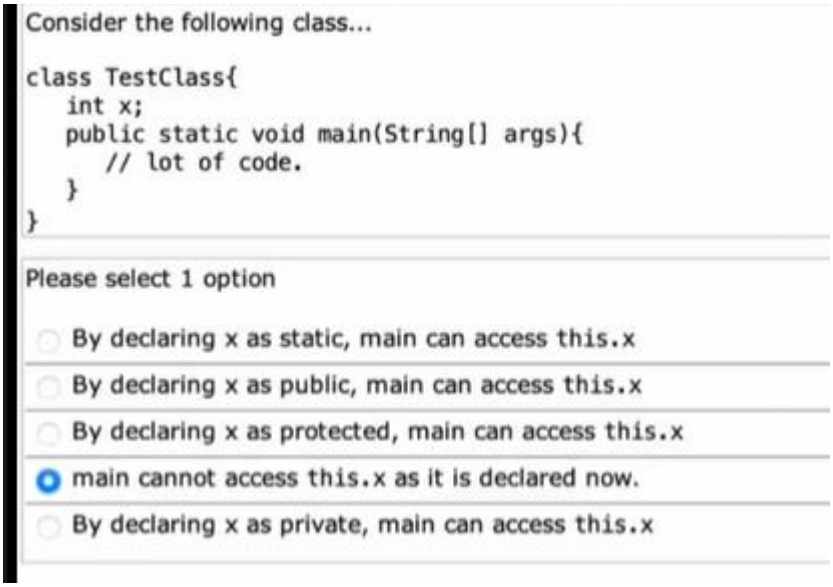
```
Bloque anonimo
Constructor Principal
Constructor Principal
Constructor Principal
```

Comienzan simuladores, Working with Constructors

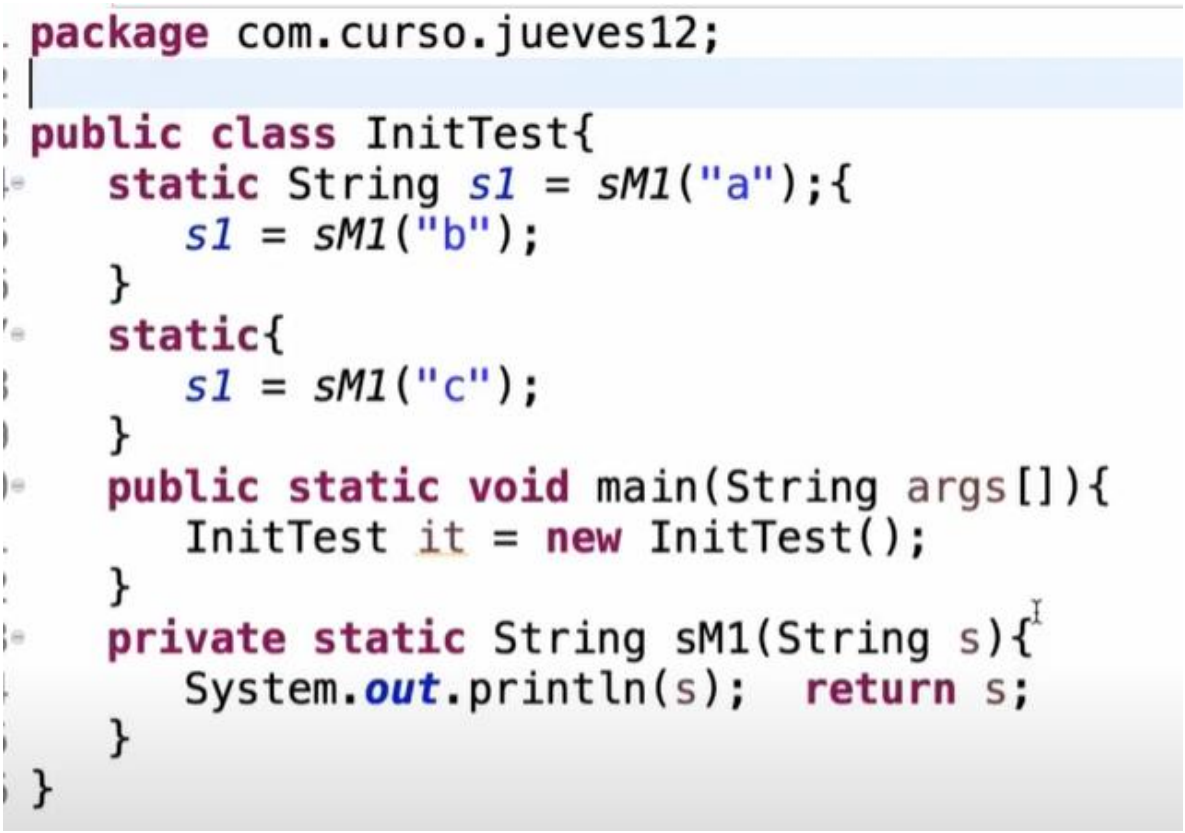


El polimorfismo hace los códigos más reusables ya que se enfoca en características grandes que pueden implementarse de acuerdo a las necesidades del código sin el hecho de generar más recursos.

Y el dinamismo permite que la decisión de qué método debe ser invocado se tome durante el *runtime* en la clase actual del objeto.



Main no puede accesar *this.x* porque main() es un método estático. Le falta el this.



```java
package com.curso.jueves12;

public class InitTest{
    static String s1 = sM1("a");{
        s1 = sM1("b");
    }
    static{
        s1 = sM1("c");
    }
    public static void main(String args[]){
        InitTest it = new InitTest();
    }
    private static String sM1(String s){
        System.out.println(s);  return s;
    }
}
```

a y c están declarados como un bloque anónimo de instancia y uno static correspondientemente, no son parte del método, por lo que estos se imprimirán primero en orden de aparición, seguidos de b, por lo que el resultado será *a, c, b.*

Consider the following code:

```
public class MyClass {

  protected int value = 10;

}
```

Which of the following statements are correct regarding the field value?

Please select 1 option

- It cannot be accessed from any other class.
- It can be read but cannot be modified from any other class.
- It can be modified but only from a subclass of MyClass.
- (●) It can be read and modified from any class within the same package.

Ya que es un valor protected, una clase fuera de este paquete que hereda MyClass solo podrá heredar el valor de la variable de la instancia de MyClass.

Given:

```
class Triangle{
    public int base;
    public int height;
    private static double ANGLE;

    public static double getAngle();

    public static void Main(String[] args) {
        System.out.println(getAngle());
    }
}
```

Identify the correct statements:

El método *getAngle();* al no ser *abstract*, requiere un comportamiento definido. Esto impedirá que el programa compile.

```java
package com.curso.jueves12;

public class TestClass{
    char c; //'NUL' = 0
    public void m1(){
        char[ ] cA = { 'a' , 'b'};
        m2(c, cA);
                                  // 0                m
        System.out.println( ( (int)c)  + "," + cA[1] );
    }
    public void m2(char c, char[ ] cA){
        c = 'b';
        cA[1] = cA[0] = 'm';
    }

    public static void main(String args[]){
        new TestClass().m1();
    }
}
```

Al ejecutarse *m1,* se define una variable *c* la cual al ser un *char* apunta a un "NUL" que equivale a 0 en entero; a pesar de que más abajo se asigna el valor de *b* a *c*, sigue conservando su valor de 0. Posterior a esto, se imprimirá el elemento *cA[1]* el cual fue definido como una *m.*

Revisión problema con uso de string y uso de concat en string

```java
package com.curso.jueves12;

public class PrincipalString {

    public static void main(String[] args) {
        String abc = "";
        abc.concat("abc");
        abc.concat("def");
        System.out.print("Resultado: "+abc);
    }

}
```

```java
package com.curso.jueves12;

public class PrincipalString {

    public static void main(String[] args) {
        String abc = ""; //Inmutable
        abc.concat("abc"); //VOLANDO EN MEMORIA
        abc.concat("def"); //VOLANDO EN MEMORIA
        System.out.print(abc);


    }

}
```

Este código no imprimirá nada, porque al ser *String* una clase inmutable, su valor ya no puede se cambiado. Sin embargo, si se pueden seguir generando *Strings* nuevos que apunten a objetos diferentes.

```java
package com.curso.jueves12;

public class PrincipalString {

    public static void main(String[] args) {
        String abc = ""; //Inmutable
        abc.concat("abc"); //VOLANDO EN MEMORIA
        abc.concat("def"); //VOLANDO EN MEMORIA
        System.out.print(abc);

        StringBuilder xyz = new StringBuilder("");
        xyz.append("ABC");
        xyz.append("DEF");
        System.out.print(xyz); //ABCDEF
```

Por su parte, el *StringBuilder* si es mutable, se le pueden añadir objetos mediante un *append*.