

# Paradigmas de la Programación – Recuperatorio del Segundo Parcial

19 de Junio de 2025

Apellido y Nombre: \_\_\_\_\_

Ej. 1

Ej. 2

Ej. 3

Ej. 4

- 1.
2. [10 pt.] Estos dos códigos, en PHP y en Ruby respectivamente, parecen muy semejantes pero se comportan de forma distinta:

PHP:

```
1 class ClassA {
2     public $myvar;
3
4     public function __construct() {
5         $this->myvar = "hello";
6     }
7
8     public function getMyVar() {
9         echo $this->myvar;
10    }
11 }
12
13 class ClassB extends ClassA {
14     public function __construct() {
15         $this->myvar = "goodbye";
16     }
17 }
18
19 $demo1 = new ClassA ();
20 $demo2 = new ClassB ();
21
22 $demo1->getMyVar ();
23 $demo2->getMyVar ();
24 $demo1->getMyVar ();
```

Ruby:

```
1 class ClassA
2     @@my_var = nil
3
4     def initialize
5         @@my_var = "hello"
6     end
7
8     def my_var
9         puts @@my_var
10    end
11 end
12
13 class ClassB < ClassA
14     def initialize
15         @@my_var = "goodbye"
16     end
17 end
18
19 demo1 = ClassA.new
20 demo1.my_var
21 demo2 = ClassB.new
22
23 demo2.my_var
24 demo1.my_var
```

El código en PHP imprime *"hello goodbye hello"*. El código en Ruby, en cambio, imprime: *"hello goodbye goodbye"*. Según este comportamiento observable, de una descripción sobre el alcance y comportamiento de la variable *myvar* en cada uno de los dos programas. Si le resulta más cómodo, puede tener en cuenta que se trata de un constructor en PHP y de una variable de clase en Ruby.

3. [10 pt.] Los dos programas siguientes tienen un comportamiento muy parecido. Explique la semántica de la palabra clave **friend** comparándola con el otro programa.

```
1  class Persona {
2  private:
3      string nombre;
4
5  public:
6      Persona(string n) : nombre(n) {}
7      friend class Amigo;
8  };
9
10 class Amigo {
11 public:
12     void mostrarNombre(const Persona& p) {
13         cout << "El_nombre_de_la_persona_es:_" << p.nombre << endl;
14     }
15 };
16
17 int main() {
18     Persona p("Luis");
19     Amigo a;
20     a.mostrarNombre(p);
21     return 0;
22 }
```

```
1  class Persona {
2  private:
3      string nombre;
4
5  public:
6      Persona(string n) : nombre(n) {}
7
8      string getNombre() const {
9          return nombre;
10     }
11 };
12
13 class Amigo {
14 public:
15     void mostrarNombre(const Persona& p) {
16         cout << "El_nombre_de_la_persona_es:_" << p.getNombre() << endl;
17     }
18 };
19
20 int main() {
21     Persona p("Luis");
22     Amigo a;
23     a.mostrarNombre(p);
24     return 0;
25 }
```

4. [10 pt.] El siguiente texto describe el comportamiento de la instrucción *final* en Java:

When you declare a variable as final, it must be initialized only once. Once a final variable has been assigned a value, attempting to modify it will result in a compile-time error. This immutable property makes final variables akin to constants.

Final methods are methods that cannot be overridden in any subclass. This is particularly useful when you need to maintain a consistent implementation of a method across various subclasses in the class hierarchy, thus avoiding unintended behaviors or security breaches.

A final class is one that cannot be subclassed. This restriction is typically employed to maintain the immutability of the class or to provide a guarantee that certain behavior is preserved without alteration through inheritance. Final classes are often used in conjunction with final variables to create fully immutable objects which are thread-safe by design.

Según estas definiciones, argumente cuáles de los siguientes códigos no compilarían y por qué.

Código A:

```
1 public class Animal {
2     public final void makeSound() {
3         System.out.println("The animal makes a sound");
4     }
5 }
6
7 public class Dog extends Animal {
8     @Override
9     public void makeSound() {
10         System.out.println("The dog barks");
11     }
12 }
```

Código B:

```
1 public final class A {
2     private final int value;
3
4     public A(int value) {
5         this.value = value;
6     }
7
8     public int getValue() {
9         return value;
10    }
11 }
12
13 public class B extends A { }
```

Código C:

```
1 public class A {
2     public static void main() {
3         final int foo = 10;
4         foo = 20;
5     }
6 }
```