# Paradigmas de la Programación – Recuperatorio del Segundo Parcial

18 de Junio de 2024

**Apellido y Nombre:** _____

Ej. 1 Ej. 2 Ej. 3 Ej. 4

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

PHP:

```php
class ClassA {
    public $myvar;

  public function __construct() {
    $this->myvar = "hello";
  }

  public function getMyVar() {
    echo $this->myvar;
  }

}
class ClassB extends ClassA {
    public function __construct() {
        $this->myvar = "goodbye";
    }
}

$demo1 = new ClassA();
$demo2 = new ClassB();

$demo1->getMyVar();
$demo2->getMyVar();
$demo1->getMyVar();
```

Ruby:

```ruby
class ClassA
  @@my_var = nil

  def initialize
    @@my_var = "hello"
  end

  def my_var
    puts @@my_var
  end
end

class ClassB < ClassA
  def initialize
    @@my_var = "goodbye"
  end
end

demo1 = ClassA.new
demo1.my_var
demo2 = ClassB.new

demo2.my_var
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.

2. **[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  }
```

3. **[10 pt.]** Teniendo en cuenta el alcance de los diferentes niveles de visibilidad en la orientación a objetos, explique qué semántica es necesario que tenga la palabra clave *friend* de la línea 9 para que el siguiente código en C++ compile sin problemas.

```
1  class ClassB; // Forward declaration of ClassB
2
3  class ClassA {
4      private:
5          int numA;
6      public:
7          ClassA() : numA(0) {}
8
9          friend class ClassB; // Friend class declaration
10
11         void display() {
12             cout << "ClassA: " << numA << endl;
13         }
14 };
15
16 class ClassB {
17     private:
18         int numB;
19     public:
20         ClassB() : numB(0) {}
21
22         void setValue(ClassA& objA, int value) {
23             objA.numA = value; // Accessing private member of ClassA
24         }
25
26         void display(ClassA& objA) {
27             cout << "ClassB: " << numB << endl;
28             objA.display(); // Accessing ClassA's member function
29         }
30 };
31
32 int main() {
33     ClassA objA;
34     ClassB objB;
35
36     objB.setValue(objA, 100);
37     objB.display(objA);
38
39     return 0;
40 }
```

4. **[10 pt.]** Explique por qué el siguiente código en Java no genera un *name clash*.

```
1  public interface Interface1 {
2
3          // create a method
4          public void show();
5  }
6
7  public interface Interface2 {
8
9          // create a method
10         public void show();
11 }
12
13 public class Case1 implements Interface1, Interface2 {
14         // implement the methods of interface
15         public void show()
16         {
17                 System.out.println("Geeks For Geeks");
18         }
19         public static void main(String[] args)
20         {
21                 // create object
22                 Case1 obj = new Case1();
23                 // using object call the implemented method
24                 obj.show();
25         }
26 }
```