Paradigmas de la Programación – Segundo Parcial

8 de Mayo de 2025

Apellido y Nombre: _____

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

1. En el siguiente diagrama se puede ver un esquema simplificado del uso de la memoria en tiempo de ejecución de un programa en C++ (izquierda) y en Smalltalk (derecha). En cada diagrama, la estructura de la izquierda representa la pila, mientras que todas las flechas y cajas a la derecha de la pila se encuentran en el *heap*.

   En ambos programas se maneja un objeto ColorPoint que es una instancia de la clase ColorPoint, que a su vez es una subclase de la clase Point. Teniendo esto en cuenta, seleccione con un círculo las expresiones verdaderas:

   a) **[2 pt.]** El programa en Smalltalk no permite reemplazo en caliente.

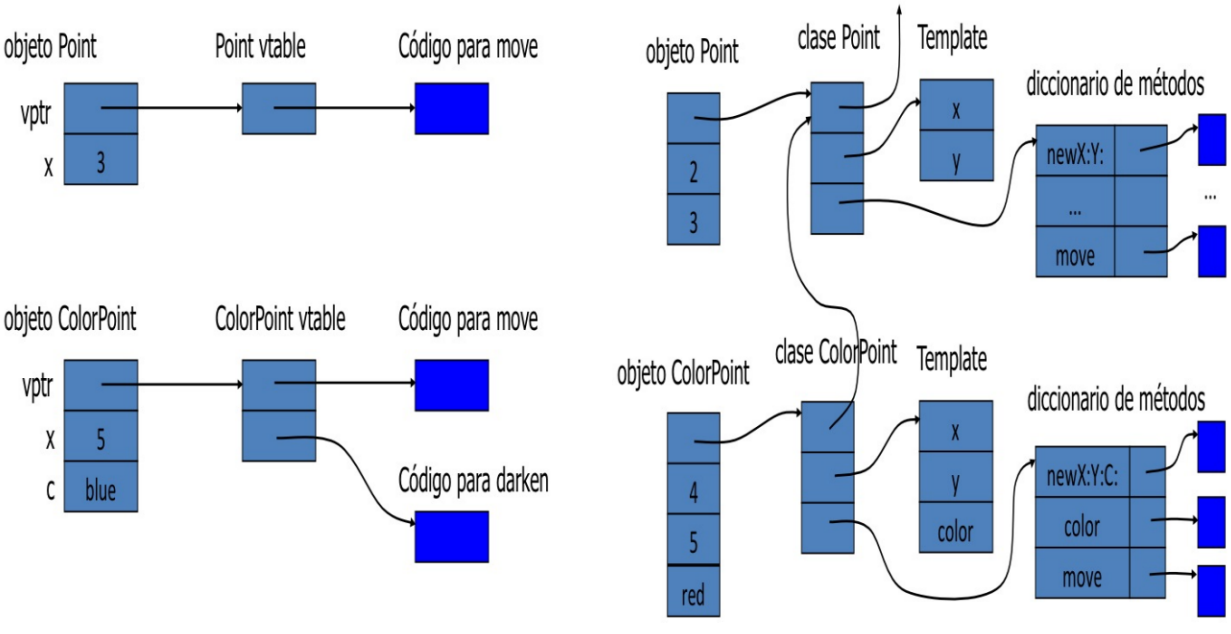
   b) **[2 pt.]** El programa en C++ (izquierda) tiene más *overhead* (es menos eficiente en tiempo de ejecución) que el programa en SmallTalk (dererecha)

   c) **[2 pt.]** El programa en C++ (izquierda) tiene menos *overhead* (es más eficiente en tiempo de ejecución) que el programa en SmallTalk (dererecha)

   d) **[2 pt.]** Las clases que heredan de Point pueden reescribir el código de la función *move* en ambos programas.

   e) **[2 pt.]** Las clases que heredan de Point pueden reescribir el código de la función *move* solamente en C++.

   f) **[2 pt.]** Las clases que heredan de Point pueden reescribir el código de la función *move* solamente en Smalltalk.



2. **[10 pt.]** En el siguiente programa en Python, describa que hace la palabra clave `super` (en 5 renglones o menos):

```python
class Persona:
    def __init__(self, nombre):
        self.nombre = nombre

    def saludar(self):
        print(f"Hola, soy {self.nombre}")

class Estudiante(Persona):
    def __init__(self, nombre, carrera):
        super().__init__(nombre)  # Llama al constructor de Persona
        self.carrera = carrera

    def saludar(self):
        print(f"Hola, soy {self.nombre} y estudio {self.carrera}")
```

3. **[10 pt.]** El siguiente programa en Java da un error:

```java
class Dato {
    private String nombre;
}

public class Main {
    public static void main(String[] main){

        Dato d = new Dato();

        d.nombre = "nombre";
        System.out.println(d.nombre);
    }
}
```

En cambio, esta otra versión (equivalente) del mismo programa no da ningún error:

```java
class Dato {
    private String nombre;

    public String getNombre() {
        return this.nombre;
    }
    public void setNombre(String nombre) {
        this.nombre= nombre;
    }
}
public class Main {
    public static void main(String[] main){
        Dato d = new Dato();

        d.setNombre("nombre");
        System.out.println(d.getNombre());
    }
}
```

Explique por qué en 5 renglones (o menos). **[10 pt.]**

4. Observe el siguiente código en Ruby, que compila e imprime "Volar" y "Nadar" (en ese orden):

```ruby
class Ave
  def hablar
    puts "Pio!"
  end
  def desplazamiento
    puts "Volar"
  end
end

class Pinguino < Ave
  def desplazamiento
     puts "Nadar"
  end
end

class PinguinoVolador < Ave; Pinguino
end

class PinguinoPinguino < Pinguino; Ave
end

pinguinoVolador = PinguinoVolador.new
pinguinopinguino = PinguinoPinguino.new

pinguinoVolador.desplazamiento
pinguinopinguino.desplazamiento
```

Indique con un círculo cuáles de las siguiente expresiones son ciertas:

a) **[2 pt.]** En Ruby, las definiciones de función son equivalentes a funciones virtuales en C++.

b) **[2 pt.]** En Ruby, las definiciones de función son equivalentes a funciones finales en Java.

c) **[2 pt.]** En Ruby, la estrategia para resolver conflictos de nombre en caso de ambigüedad (*name clashes*) consiste en prefijar el nombre de la clase ancestro de la cual se quiere heredar la implementación, prefijándolo al nombre de la función que presenta la ambigüedad.

d) **[2 pt.]** El principio de Liskov (por Bárbara Liskov) dice que "*Una subclase debe poder ser utilizada en lugar de su clase base sin alterar el comportamiento esperado del programa.*". En este ejemplo de Ruby, la clase `pinguinoVolador` no mantiene el principio de Liskov porque su comportamiento no es consistente con el de su clase madre (presenta semántica diferente para las mismas palabras).

e) **[2 pt.]** En este ejemplo de Ruby, la clase `pinguinopinguino` no mantiene el principio de Liskov porque su comportamiento no es consistente con el de su clase madre (presenta semántica diferente para las mismas palabras).