Paradigmas de la Programación
Práctico 6: Programación Orientada a Objetos (3)
algunos detalles más

Laura Alonso Alemany Ezequiel Orbe

25 de mayo de 2015

Seleccione la(s) respuesta(s) correcta(s)

1. Los objetos se pueden considerar como...
   a) instancias de clases.
   b) ejecuciones de una función que es la clase.
   c) activation records de una clase.
2. Muchos lenguajes proveen sintaxis amigable para referirse a algunas clases dentro de la jerarquía de clases, que son:
   a) las clases hermanas de la clase desde las que se las refiere.
   b) la clase madre de la clase desde que se la refiere.
   c) las clases ancestro de la clase desde la que se está trabajando.
   d) la propia clase desde la que se está trabajando.
   e) las clases descendientes de la clase desde la que se está trabajando.

Ejercicios prácticos

1. Reescriba el siguiente código en Java en Haskell, de forma que su semántica denotacional sea idéntica.

```
import core.List;
import static core.List.*;

import java.util.NoSuchElementException;

class P01 {

    <T> T f1(List<T> list) {
        if (list.isEmpty()) throw new
            NoSuchElementException("List is empty");
        List<T> elements = list.tail();
        List<T> result = list;
        while (elements.nonEmpty()) {
            result = elements;
            elements = elements.tail();
        }
```

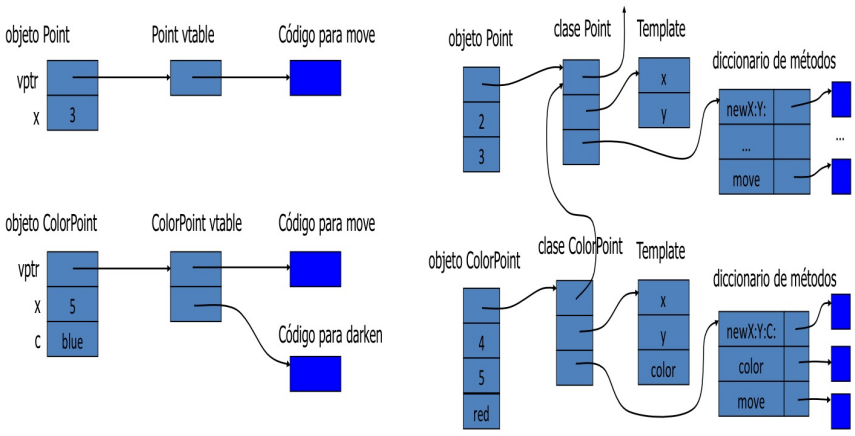
```
        return result.head();
    }
```

2. El siguiente ejemplo en C++ causa un error de compilación. Por qué? cómo puede solucionarse?

```
class trabajador
{
  public:
     void hora_de_levantarse( )
     { .... ... .... }
};
class estudiante
{
    void hora_de_levantarse( )
     { .... ... .... }
};
class ayudante_alumno : public trabajador, public estudiante
{

};

int main()
{
    ayudante_alumno obj;

    obj.hora_de_levantarse( )
}
```

3. Explique qué tests realizaría para saber cuál es la visibilidad de un método *protected* en Ruby.

4. Explique por qué Smalltalk es menos eficiente que C++ utilizando el concepto de *overhead*. Puede ayudarse de los siguientes gráficos, en los que se muestra el proceso de lookup de información asociada a los objetos en uno y otro lenguaje.



5. En el siguiente código el compilador está creando una instancia de una subclase anónima de la clase abstracta, y luego se está invocando al método de la clase abstracta. Explique por qué es necesario que el compilador haga este proceso.

```
abstract class my {
    public void mymethod() {
        System.out.print("Abstract");
    }
}

class poly {
    public static void main(String a[]) {
        my m = new my() {};
        m.mymethod();
    }
}
```

6. En el siguiente código java, inserte en el `main` los mecanismos de manejo de excepciones necesarios para capturar la excepción de forma adecuada.

```
public class BankDemo
{
   public static void main(String [] args)
   {
      CuentaBancaria c = new CuentaBancaria(101);
      System.out.println("Depositando $500...");
      c.depositar(500.00);
      System.out.println("\nRetirando $100...");
      c.retirar(100.00);
      System.out.println("\nRetirando $600...");
      c.retirar(600.00);
    }
}
```

```
public class CuentaBancaria
{
   private double balance;
   private int numero;
   public CuentaBancaria(int numero)
   {
      this.numero = numero;
   }
   public void depositar(double cantidad)
   {
      balance += cantidad;
   }
   public void retirar(double cantidad) throws
      ExcepcionFondosInsuficientes
   {
      if(cantidad <= balance)
      {
         balance -= cantidad;
      }
      else
```

```
      {
         double necesita = cantidad - balance;
         throw new ExcepcionFondosInsuficientes(necesita);
      }
   }
   public double getBalance()
   {
      return balance;
   }
   public int getNumber()
   {
      return numero;
   }
}
```