Trabajo Práctico N° 09

Objetos

Recomendaciones Generales e Información

- Conceptos relacionados a Objetos

Alcance por uso	Relación en el árbol de anidamiento entre una unidad o método y un método “m”. Esta relación existe porque la unidad o método declara un objeto que posee un método “m” con un modificador de tipo “público” y sin modificador “static”
Invocación a métodos	Se pasa por referencia el objeto desde el cual se llama al método y el método. De forma implícita en la mayoría de los lenguajes.
Almacenamiento	Los objetos son variables, por lo tanto en general pueden estar almacenados en las mismas ubicaciones que las variables (C++, Delphi, etc.). Algunos lenguajes almacenan los objetos únicamente como variables dinámicas o como variables estáticas (Java).
Herencia de atributos	Los atributos heredados de una clase “A” se ubican junto a los atributos del objeto de la clase “B” que hereda de “A”. Si se sobreescribe un atributo en B, en el objeto de la clase B habrá dos juegos de atributos.
Herencia de métodos	Los métodos heredados de una clase “A” se ubican en la Tabla de Despacho de A. Si una clase B hereda de A y sobreescribe algunos métodos, estos estarán en la Tabla de Despacho de la clase B junto con los métodos que solamente son de B. La Tabla de Despacho de la clase B contiene además por lo general un puntero a la Tabla de Despacho de la clase A para utilizar los métodos heredados y no sobreescritos.
Acceso del método al objeto	El método posee un parámetro formal que es implícito en la mayoría de los lenguajes. En algunos además se puede utilizar explícitamente con los identificadores **this** (C++, Java, C#, PHP), **self** (Delphi, Perl, Python, Ruby, Swift) o **Me** (Visual Basic). Unos pocos lenguajes poseen este parámetro únicamente de forma explícita (Python, Go, Rust)
Invocación estática	Cuando los métodos no se declaran como “virtuales” (C++ por defecto o Java con el modificador static), la invocación al método se resuelve en compilación.
Invocación dinámica (late binding call)	Si los métodos se declaran como “virtuales” (modificador “virtual” en C++ o por defecto en Java), un objeto al que se le asigna otro objeto de una clase que sobreescribe un método, puede llamar al método sobreescrito.
Objetos en lenguajes dinámicos	En los lenguajes dinámicos al poder cambiar de tipo, cuando a un objeto se le asigna otro, por lo general adquiere los atributos y métodos de la clase del segundo, sin importar si es de una clase relacionada o no. Algunos lenguajes Dinámicos con verificación de tipos actúan de la misma manera que los lenguajes compilados en el sentido que solo es posible asignar un objeto a otro que cumpla la relación “es un”.

Se recomienda utilizar los compiladores: **gcc** versión 4 o superior para **C++** (versión 99 o superior), **Java** versión 7 y **Python** versión 3.6 o superior.

Estos compiladores pueden encontrarse online (dependiendo de la disponibilidad) en plataformas como por ejemplo:

- https://www.onlinegdb.com/
- https://www.jdoodle.com/
- https://ideone.com/
- https://paiza.io/
- https://repl.it/
- https://www.tutorialspoint.com/compile_cpp_online.php
- https://www.pythonanywhere.com/

Ejercicio 1

Alcance convencional y Alcance por uso

Dado el siguiente programa en el lenguaje **C++**:

```
class A {
   public:
   int x=1;
   void m() {
   x = 1;
   fun1();
   }
};
class B {
   public:
   A *e;
   void s() {
   int c=2;
   e = new A();
   A f;
   f.m();
   }
};
void fun1() {
 B g;
 g.e = new A();


}
int main() {
   static A y;
   A z;
   A *w = new A();
   {
       B *r = new B();
       r->s();
   }
}
```

a) Realizar el árbol de anidamiento indicando donde exista alcance convencional y alcance por uso
b) Realizar un diagrama donde se muestre la ubicación de las variables en memoria
c) Clasificar cada variable según su almacenamiento
d) Ordenar las variables de forma decreciente según su tiempo de vida

Ejercicio 2

Dado el siguiente programa en **C++**

```
#include <iostream>
using namespace std;
static int z;
class A {
 public: int x;
 static int y;
};
int A::y = 1;
int main() {
 int w;
 A a1;
 A *a2 = new A();
 int *s = (int *) malloc(sizeof(int));
 printf("dirección de w: %d \n",&w);
 printf("dirección de z: %d \n",&z);
 printf("dirección de lo apuntado por s: %d \n",&(*s));
 printf("dirección de a1: %d \n",&a1);*a2
 printf("dirección de a1.x: %d \n",&a1.x);
 printf("dirección de a1.y: %d \n",&A::y);
 printf("dirección de a2: %d \n",&a2);
 printf("dirección de lo apuntado por a2: %d \n",&(*a2));
 printf("dirección del componente x de lo apuntado por a2: %d \n",&a2->x);
 printf("dirección del componente x de lo apuntado por a2: %d \n",&(*a2).x);
}
```

Responda las siguientes preguntas:

a) ¿Qué puede deducir de las direcciones de **w**, **a1** y **a2** respecto al almacenamiento de las variables?
b) ¿Qué puede deducir de las direcciones de **z** y **A::y** respecto al almacenamiento de las variables?
c) ¿Qué puede deducir de las direcciones de lo apuntado por **a2** y por **s** respecto al almacenamiento de las variables?
d) ¿Qué puede deducir de las direcciones de **a1** y **a1.x** respecto al almacenamiento de las variables?
e) ¿Qué puede deducir de las direcciones de lo apuntado por **a2** y ***a2.x** respecto al almacenamiento de las variables?
f) Proporcionar una explicación para el resultado de las últimas dos impresiones del programa
g) Si se crea una clase llamada **B**, idéntica a la clase **A** y se agregan las instrucciones: "**B b1 = a1**;" ¿qué puede decir de la compatibilidad de tipos para los objetos?

Ejercicio 3

Dado el siguiente programa en el lenguaje **Java**:

```
class A{
static public int x=1;
public int y=3;
public void m1() {
      int z=2;
      x++;
      y++;
   }
}

class B{
static A a2;
public void m2() {
       a2 = new A();
       System.out.println(a2);
       A a3 = new A();
       a3.m1();
     }
}
public class ejercicio3 {
 public static void main(String[] args)  {
     A a1 = new A();
     A a2 = new A();
     a1.m1();
     System.out.println(a1.x);
     System.out.println(a2.x);
     System.out.println(a2.y);
     B b1 = new B();
     B b2 = new B();
     b1.m2();
     System.out.println(a1);
     System.out.println(b1);
     System.out.println(b2);
     System.out.println(b1.a2);
     System.out.println(b2.a2);
     System.out.println(B.a2);
     b1.a2.x = 5;
     System.out.println(b2.a2.x);
}
}
```

Responda las siguientes preguntas:

a) Realizar la pila de ejecución del programa

b) ¿Qué puede deducir del valor de las impresiones 4, 8, 9 y 10?

c) ¿Qué sucede si se cambia el valor del atributo x del atributo a2 de los objetos de la clase B?

Ejercicio 4

Dado el siguiente programa en el lenguaje **C++**:

```
#include <stdio.h>
#include <new>
class Padre {
   public: int x=1;
   int z=4;
class Hija : public Padre{
   public: int x=2;
   int y=3;
};
int main(){
   Padre p1 = Padre();
   Hija h1 = Hija();
   h1.x = 3;
   printf("&p1 %d\n",&p1);
   printf("&h1 %d\n",&h1);
   p1 = h1;  //1
   printf("&p1 %d\n",&p1);
   printf("p1.x %d\n",p1.x);
   printf("p1.y %d\n",p1.y);
   printf("h1.x %d\n",h1.x);
   printf("h1.y %d\n",h1.y);
   printf("&p1.x %d\n",&p1.x);
   printf("&h1.x %d\n",&h1.x);
   p1.m();
   h1 = p1;
   return 0;
}
```

Responda las siguientes preguntas:

a) Indicar cuáles instrucciones provocan errores, qué tipo de errores, a qué se deben y cómo se podrían solucionar si es que esto es posible.

b) ¿Cuál es la semántica de la asignación de la línea `//1`?

c) ¿Cuáles cambios semánticos se producen si se reemplaza la asignación por "`p1 = (Padre)h1;`"?

d) ¿Por qué la dirección del atributo "**x**" coincide con la del objeto **p1**, mientras que eso mismo no sucede con el atributo "**x**" del objeto **h1**?

Ejercicio 5

Dado el siguiente programa en el lenguaje **Java**:

```
class Padre {
   public int x=1;
   int y= 2;
   Padre() {};
   void m1() {
      System.out.println("x en m1 Padre:" + x);
   }
   void m2() {
      System.out.println("-x en m2 Padre:" + -x);
   }
   void printthis() {
       System.out.println("this: " + this);
   }
};
class Hija extends Padre {
   int x=3;
   int z=4;
   void m1() {
      System.out.println("-x en m1 Hija: " + -x);
   };
   void m3() {
      System.out.println("x en m3 Hija: " + x);
   }
   void printthis() {
       System.out.println("this: " + this);
   }
};
public class ejercicio6{
   public static void main(String []args){
       Padre p1 = new Padre();
       Hija h1 = new Hija();
       System.out.println("p1.x: " + p1.x);
       System.out.println("p1.y: " + p1.y);
       System.out.println("h1.x: " + h1.x);
       System.out.println("h1.y: " + h1.y);
       System.out.println("h1.z: " + h1.z);
       p1.m1();
       p1.m2();
       h1.m1();
       h1.m2();
       h1.m3();
       p1.printthis();
       h1.printthis();
       p1 = h1; //1
       //h1 = p1; //2
       System.out.println("Después de asignación");
       System.out.println("p1.x: " + p1.x);
       System.out.println("p1.y: " + p1.y);
       System.out.println("h1.x: " + h1.x);
       System.out.println("h1.y: " + h1.y);
       System.out.println("h1.z: " + h1.z);
       p1.m1();
       p1.m2();
       h1.m1();
       h1.m2();
       h1.m3();
       p1.printthis();
       h1.printthis();
   }
}
```

a) Realizar un diagrama de memoria donde se muestre la ubicación de los objetos y los efectos de las asignaciones.
b) ¿Cuál es la semántica de la instrucción comentada con `//1` y cuáles diferencias observa con el ejemplo en **C++**?
c) ¿Qué puede deducir de los resultados de las impresiones?
d) ¿Qué ocurre si se reemplaza la instrucción comentada con`//1` por la instrucción comentada con `//2`?
e) ¿Qué ocurre si se reemplaza la instrucción comentada con `//2` por la instrucción: “h1 = (Hija) p1”?

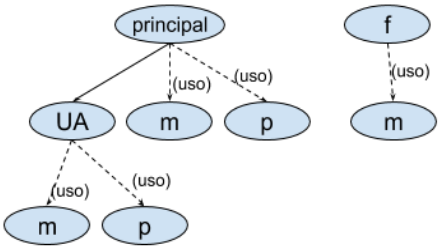
Ejercicio 6

Dado el siguiente programa en un lenguaje en el cual todos los atributos y métodos de una clase son públicos por defecto y posee compatibilidad por nombre para clases y punteros.

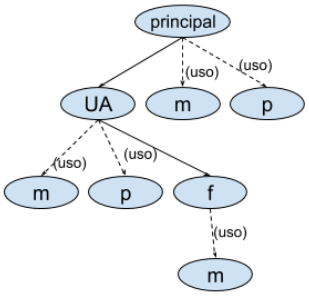
```
clase A {
        w=1
        x=2
        estatica y
        func m() {
              imprime &w , &x , &y
        }
}

clase B hereda de A {
        x=4
        z=5
        func p() {imprime &w , &x , &y , &z}
}

A::y = 3
func f() {
    estatica A a1
 }

func principal()
{
  A a2
  A *a3 = nuevo A()
  B *b1
  {
   estatica B b2
   b1 = nuevo B()
   f()
   // impresiones
   a2.m()
   a3->m()
   b1->m()
   b1->p()
  }
  // Asignaciones
}
```

a) Seleccionar un árbol de anidamiento correcto (árbol de anidamiento representado como una lista de arcos de anidamiento y el tipo de alcance.

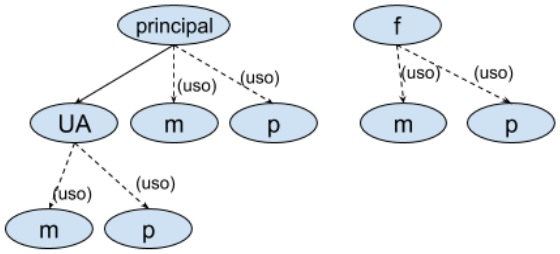
Opción1:



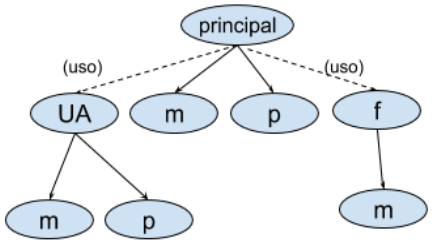
Opción 2:



Opción 3:



Opción 4:



b) Si se insertan por separado cada una de las siguientes asignaciones en la línea comentada con "// Asignaciones", responder cuáles son los tipos que el lenguaje detecta a izquierda y a derecha, si la asignación provoca algún tipo de error y la semántica de la asignación.

Nro.	Asignación	Izq	Der	Error	Semántica
1	a2=a3				
2	*b1=*a3				
3	a2=(A) *b1				
4	a3=a2				
5	a3= (A*) b1				
6	b1=a3				
7	a3=*b1				
8	*a3=a2				
9	*a3=b1				
10	a2=b1				
11	b1=*a3				
12	*b1=a2				
13	*a3=(A) *b1				
14	*b1=a3				
15	a2=*a3				
16	b1=a2				

c) Idem ejercicio b) con las siguientes asignaciones:

Nro.	Asignación	Izq	Der	Error	Semántica
1	*a2 = a3				
2	*a2=b1				
3	*a2=*a3				
4	*a2=*b1				
5	a3=*a2				
6	*a3=*a2				
7	b1=*a2				
8	*b1=*a2				

d) Responder qué sucede si se reemplazan las siguientes instrucciones, quitando las conversiones explícitas.

Antes	Ahora	Funcionaba antes y sigue funcionando igual	Antes daba error de compilación y ahora no	Antes daba error de compilación y ahora también
a2=(A) *b1	a2= *b1	SI/NO	SI/NO	SI/NO
a3= (A*) b1	a3= b1	SI/NO	SI/NO	SI/NO
*a3=(A) *b1	*a3= *b1	SI/NO	SI/NO	SI/NO

e) En el siguiente modelo de memoria del programa original (sin información de cadenas estática y dinámica), indicar qué contiene cada dirección y el valor.

Dirección	Descripción	Valor
-$50000		
-$50004		
-$50008		
-$50012		
-$50016		
-$50020		
-$50024		
$100		
$104		
$108		
$112		
$120		
$1000		
$1004		
$4000		
$4004		
$4008		
$4012		

Ejercicio 7

Dado el siguiente programa en el lenguaje **Python**:

```
class Padre:
    x = 1
    y = 2
    def m1(self):
        print (self.x)
    def m2(self):
        print (-self.x)

class Hija(Padre):
    x = 3
    z = 4
    def m1(self):
        print (-self.x)
    def m3(self):
        print (self.x)
p1 = Padre()
h1 = Hija()
print ("antes de asignación")
p1.m1()
p1.m2()
h1.m1()
h1.m2()
h1.m3()
p1 = h1 #1
h1 = p1 #2
print ("después de asignación")
p1.m1()
p1.m2()
h1.m1()
h1.m2()
h1.m3()
p1.m3()
```

a) Realizar un diagrama de memoria donde se muestre la ubicación de los objetos y los efectos de las asignaciones.
b) Quitar el comentario de en las líneas #1 y #2 de a una por vez y analizar los resultados de la ejecución.
c) ¿Cuáles diferencias semánticas se observan en la invocación al método **m2** respecto de **Java** y **C++**?

Ejercicio 8

Dados los siguientes códigos en lenguajes **C++** y **Java**. Considerar que se usa garbage collection con el método de contadores de referencia y el algoritmo first fit cuando se pide memoria.

C++

```
Class A{
  int x;
  A(int num){
     x=num;}   [1]
}
Class B{
  int x;
  B(int num){
     x=num;}
}
Class C{
  A* a;
  C(A* aC){
     a=aC;}   [2]
}
int main(){
  A a1(1);
  A a2(5);
  B b1(8);
  C c1(&a1);
  a2=a1;     [3]
}
```

Java

```
Class A{
  public int x;
  A(int num){
     x=num;}    [1]
}
Class B{
  public int x;
  B(int num){
     x=num;}
}
Class C{
  public A a;
  C(A aC){
     a=aC;}     [2]
}
public class Programa{
public static void main(String []args){
  A a1= new A(1);
  A a2= new A(5);
  B b1= new B(8);
  C c1= new C(a1);
  a2=a1;         [3]
}}
```

Responda a cada inciso *para cada lenguaje por separado*, o indique de una misma forma si ambos lenguajes se comportan igual.

a) Realizar un esquema de memoria, uno para cada lenguaje, para ambas ejecuciones.
b) Clasifique todas las variables de acuerdo a su almacenamiento.
c) Indicar para las asignaciones [1] [2] y [3], si se hacen por copia o por referencia.
d) Analice si, utilizando los objetos creados (a1, a2, b1, c1) y sin crear ningún objeto nuevo, se pueden producir las siguientes situaciones:
   - Punteros colgados
   - Fragmentación
   - Garbage
e) En caso de ser posible, indique las instrucción/es que lo generarían. Si no es posible, explicar por qué no lo es.
f) La siguiente afirmación

> La variable c1 se puede convertir en garbage ya que tiene un puntero a un objeto de la clase A como atributo

¿Es verdadera o falsa? Justifique

g) Analice, para cada lenguaje, si los objetos b1 y a1 son *compatibles por estructura y si son compatibles por nombre.*
¿Es posible realizar la asignación b1=a1 en alguno de los lenguajes? **Justificar sus respuestas.**