Ejercicios LTP - SOLUCIONES

Tema 1: Introducción

Parte I: Cuestiones

1. Los siguientes programas Java contienen errores. Señala las sentencias que contienen errores. Explica en qué consiste cada error y cómo podría solucionarse.

Programa 1:

```
public class Exercise1 {
  public static void main(String[] args) {
    A objA = new A();
    System.out.println("in main(): ");
    System.out.println("objA.a ="+objA.a);
    objA.a = 222;
  }
}
```

```
public class A {
    private int a = 100;
    public void setA(int value) {
       a = value;
    }
    public int getA() {
       return a;
    }
}//class A
```

En el método main se intenta acceder a un atributo privado de A
Solución. Se deben usar los métodos definidos para el acceso al atributo privado. Las dos últimas líneas de main se reemplazarán por:

```
System.out.println("objA.a = " + objA.getA());
objA.setA(222);
```

Salida del programa:

```
in main():
objA.a = 100
```

Programa 2:

```
public class Exercise2 {
  public static void main(String[] args) {
    System.out.println("in main(): ");
    System.out.println("objA.a ="+getA() );
    setA(123);
  }
}
```

```
public class A {
    private int a = 100;
    public void setA(int value) {
       a = value; }
    public int getA() {
       return a; }
}//class A
```

Se quieren usar métodos definidos en la clase A sin haber creado un objeto.
Error: Cannot find symbol `getA()`

Solución:

```
public class Exercise2{
  public static void main(String[] args) {
    A objA = new A();
    System.out.println("in main():  ");
    System.out.println("objA.a = "+objA.getA() );
    objA.setA(123)}
}
```

Salida del programa:

```
in main():
objA.a = 100
```

2. Dado el siguiente programa, señala el (o los) error(es) que pueda haber en las últimas cuatro asignaciones de la clase `PolymorphicAssignment`.

```
public class ClassA {
}// end ClassA

public class ClassB extends ClassA {
}// end ClassB
```

```
public class PolymorphicAssignment {
  public static void main(String[] args) {
    ClassA obj1 = new ClassA();
    ClassA obj2 = new ClassA();
    ClassB obj3 = new ClassB();

    obj1 = obj2;
    obj1 = obj3;
    obj3 = obj2;
    obj3 = obj1;
  } //end main
} //end class
```

Primera asignación: Ningún problema aquí. `obj1` y `obj2` son del mismo tipo de datos (`ClassA()`)

Segunda asignación: No hay problema porque `obj3` es una variable referencia del objeto de tipo `ClassB`, y los objetos de `ClassB` también pertenecen a `ClassA`.

Tercera asignación: Esta asignación produce error ya que está intentando asignar un objeto de `ClassA` a una variable de la subclase `ClassB`. Mensaje de compilación: "incompatible types"

Cuarta asignación: También incompatible ya que el objeto referenciado en `obj1` es visto como `ClassA` aunque desde `obj3` sea visto como de `ClassB`.

3. ¿Verdadero o falso?:

   (a) Una subclase es generalmente más pequeña que su superclase. FALSO

   (b) Un objeto de una subclase es también un objeto de su superclase. CIERTO

   (c) Todos los métodos en una clase abstracta deben declarase abstractos. FALSO

   (d) En un programa Java no puede haber dos métodos con el mismo nombre. FALSO

   (e) Una subclase de una clase abstracta que no implementa todos los métodos abstractos que hereda debe ser abstracta. CIERTO

   (f) Todos los métodos en una superclase abstracta tienen que ser abstractos. FALSO

4. Dado el siguiente programa Java, indica qué tipo/s de polimorfismo se usa/n en cada caso. Los casos están identificados por números. Cuando dos fragmentos de código tienen el mismo número es porque es necesario mirar las dos partes para determinar el tipo de polimorfismo.

```
public class Polymorphism {

//*************************************
// 5. INCLUSION
//*************************************
  abstract class Car {
    public void getBrand(){};
  }

  class Polo extends Car {
    String brand = "Volkswagen";
    public void getBrand(){
      System.out.println();
      System.out.println(brand);
    };
  }

  public void carBrand(Car c){
    c.getBrand();
  }

  public static void main(String[] args){
    Polymorphism o = new Polymorphism();

//*************************************
// 1. SOBRECARGA (de "+" y de "println")
//*************************************
    System.out.println("Hello "+"world");
    System.out.println(1+2);
    System.out.println();

//*************************************
// 2. SOBRECARGA (de "suma") y COERCION (x2=1)
//*************************************
    int x1=1,y1=1;
    o.add(x1,y1);
    float x2=1,y2=1;
    o.add(x2,y2);

//*************************************
// 3. COERCION (de int a float en los argumentos)
//*************************************
    int x3=2,y3=1;
    o.subtract(x3,y3);

//*************************************
// 4. GENERICIDAD (de "printarray")
//*************************************
    // Create arrays of Integer
    // Double and Character
    Integer[] intArray = {1,2,3,4,5};
    Double[] doubleArray = {1.1,2.2,3.3};
    Character[] charArray = {'H','E','Y'};

    System.out.println("intArray contains:");
    o.printArray(intArray);

    System.out.println("\ndoubleArray contains:");
    o.printArray(doubleArray);

    System.out.println("\ncharArray contains:");
    o.printArray(charArray);

//*************************************
// 5. INCLUSION
//*************************************
    Polo m = o.new Polo();
    o.carBrand(m);
  } // END MAIN

//*************************************
// 2. SOBRECARGA (de "suma") y COERCION (x2=1)
//*************************************
  public void add (int x, int y) {
    System.out.println("Integer addition: ");
    System.out.println(x+y);
    System.out.println();
  }

  public void add (float x, float y) {
    System.out.println("Double addition: ");
    System.out.println(x+y);
    System.out.println();
  }

//*************************************
// 3. COERCION (de int a float en los argumentos)
//*************************************
  public void subtract (float x, float y) {
    System.out.println("Double subtraction: ");
    System.out.println(x-y);
    System.out.println();
}

//*************************************
// 4. GENERICIDAD (de "printarray")
//*************************************
  public <E> void printArray(E[] inputArray) {
    for (E element : inputArray){
        System.out.printf("%s ", element);
    }
    System.out.println();
  }
} // END CLASS Polymorphism
```

Output:

```
Hello world
3

Integer addition:
2

Double addition:
2.0

Double subtraction:
1.0

integerArray contains:
1 2 3 4 5

doubleArray contains:
1.1 2.2 3.3

characterArray contains:
H E Y

Volkswagen
```

5. ¿Qué tipo de polimorfismo está presente en el siguiente fragmento de código?

```
public int  add(int x, int y){
    return x + y;
}
public String  add(String s, String t){
    return s + t;
}
```

Solución: Sobrecarga

6. ¿Verdadero o falso?:
   (a) Todos los lenguajes de programación permiten la reflexión. FALSO
   (b) La reflexión puede usarse para observar y modificar el programa en tiempo de ejecución. CIERTO
   (c) Los lenguajes orientados a objetos no soportan la reflexión. FALSO
   (d) La reflexión es una estrategia clave para el polimorfismo. FALSO

7. Tras ejecutar este código, la variable "x" vale 2 y la variable "y" vale 2. ¿Qué mecanismo de paso de parámetros se ha utilizado?

```
int funcion(int a, int b) {
    a++;
    return b;
}
int x = 1, y = 2;
y = funcion(x, 2*x);
```

Solución: Por referencia
Si fuera por valor, x valdría 1 e y valdría 2

8. Dado el siguiente programa:

```
static void foo(int a, int b) {
  a = a+b;
}
public static void main () {
  int x = 0
  int y = 10
  foo(x,y)
}
```

indica cuál es el valor de las variables x e y al terminar la ejecución de main siguiendo:

(a) la estrategia de paso de parámetros por referencia.

x=10, y=10

(b) la estrategia de paso de parámetros por valor.

x=0, y=10

9. Dado el siguiente programa:

```
public class Exercise9 {
    public static void main(String[] args) {
       CallByAny cbn = new CallByAny();
       int x = 2, y;
       y = cbn.foo(x, x+1);
       System.out.println("x: " + x);
       System.out.println("y: " + y);
    }
}
```

```
class CallByAny {
     int foo(int a, int b){
         a++;
         return (a+b);
     }
}//class CallByAny
```

(a) Teniendo en cuenta que Java utiliza un mecanismo de paso de parámetros por valor (*call by value*), indique el resultado de la ejecución de este programa.

`x: 2`
`y: 6`
Los parámetros formales (`a` y `b`) toman los valores de los parámetros reales (`2` y `3`, respectivamente) y se ejecuta el método. Trabajan sobre direcciones de memoria diferentes por lo que los parámetros reales no se modifican.

(b) ¿Cuál sería el resultado si el mecanismo de paso de parámetros fuera por referencia (*call by reference*)?

`x: 3`
`y: 6`
Los parámetros formales (`a` y `b`) se reemplazan por referencias a las direcciones de memoria de los parámetros reales (sólo para `x` ya que `x+1` es una expresión que se evalúa en el momento de la llamada). El parámetro `x` cambia de valor durante la ejecución del método.

10. Dado el siguiente programa:

```
public class Exercise10 {
    public static void main(String[] args) {
       CallByAny cba = new CallByAny();
       int x = 3, y;
       y = cba.foo(x, x+2);
       System.out.println("x=" + x);
       System.out.println("y=" + y);
    }
}
```

```
class CallByAny {
     float foo(int a, int b){
         a +=10;
         return b;
     }
}//class CallByAny
```

(a) Teniendo en cuenta que Java utiliza un mecanismo de paso de parámetros por valor (*call by value*), indique el resultado de la ejecución de este programa.

`x= 3`
`y= 5`
Los parámetros formales (`a` y `b`) toman los valores de los parámetros reales (`3` y `5`, respectivamente) y se ejecuta el método. Trabajan sobre direcciones de memoria diferentes por lo que los parámetros reales no se modifican.

(b) ¿Cuál sería el resultado si el mecanismo de paso de parámetros fuera por referencia (*call by reference*)?

```
x= 13
y= 5
```

Los parámetros formales (`a` y `b`) se reemplazan por referencias a las direcciones de memoria de los parámetros reales (sólo para `x` ya que `x+2` es una expresión que se evalúa en el momento de la llamada).

11. Dado el siguiente programa:

```
public class Clase1 {
    public static void main(String[] args){
        Example ex = new Example();
        int x = 3, y;
        y = ex.foo(x, x+2);
        System.out.println("x=" + x);
        System.out.println("y=" + y);
        }
}
```

```
class Example{
        int foo(int a, int b){
             a+=b;
        return a;
        }
}
```

(a) Teniendo en cuenta que Java utiliza un mecanismo de paso de parámetros por valor (*call by value*), indique el resultado de la ejecución de este programa.

```
x= 3
y= 8
```

(b) ¿Cuál sería el resultado si el mecanismo de paso de parámetros fuera por referencia (*call by reference*)?

```
x= 8
y= 8
```

12. Dado el siguiente programa:

```
public class Clase {

  public static void main(String[] args){
    CallByName cbn = new CallByName();
    float x = 3, y;
    y = cbn.metodo(x, 1/x);
    System.out.print("y=" + y);
    System.out.println("x=" + x);
  }
}
```

```
class CallByName{
  float metodo(float cont, float incr){
    float sum = 0;
    for (cont=1;cont<=3;cont++){
        sum = sum + incr;
    }
    return sum;
  }
}
```

(a) Teniendo en cuenta que Java utiliza un mecanismo de paso de parámetros por valor (*call by value*), indique el resultado de la ejecución de este programa.

```
y: 1
x: 3
```

Los parámetros formales (contador e incremento) toman los valores de los parámetros actuales (3 y 1/3, respectivamente) y se ejecuta el método.

(b) ¿Cuál sería el resultado si el mecanismo de paso de parámetros fuera por referencia (*call by reference*)?

```
y: 1
x: 4
```

Los parámetros formales (contador e incremento) se reemplazan por referencias a las direcciones de memoria de los parámetros actuales (solo para x ya que 1/x es una expresión que se evalúa en el momento de la llamada).

13. ¿Verdadero o falso?

    (a) En el alcance estático el ámbito de una variable es el fragmento de código sintácticamente más próximo a su declaración. CIERTO

    (b) El recolector de basura (*garbage collector*) se encarga de la asignación y liberación automática de los recursos de memoria para un programa. CIERTO

14. Dado el siguiente programa:

```
1    program scoping
2    var x: integer;
3    procedure one;
4     var x: integer;
5     begin
6        x:=13;
7        three;
8     end;//one
9    procedure two;
10    var x: integer;
11    begin
12       x:= 12;
13       three;
14    end;//two
15   procedure three;
16    begin
17       writeln(x);
18    end;//three
19   begin
20      x:= 14;
21      one;
22      two;
23   end.
```

    (a) ¿Cuál es el resultado de la ejecución del programa `scoping` considerando alcance estático?

    14
    14
    La función `three` no tiene a `x` como variable local, por lo que el parámetro `x` en la línea `17` se refiere a la variable global `x`.

    (b) ¿Cuál es el resultado de la ejecución del programa `scoping` considerando alcance dinámico?

    13
    12
    Cuando después de la llamada a `one` (línea `21`), ésta llama a `three` (línea `7`), la variable `x` en la línea `17` se refiere a la variable local `x` definida en `one` (línea `4`) e inicializada al valor `13` (línea `6`). Análogamente, cuando después de la llamada a `two` (línea `22`), ésta llama a `three` (línea `13`), la variable `x` en la línea `17` se refiere a la variable local `x` definida en `two` (línea `10`) e inicializada al valor `12` (línea `12`).

15. Dado el siguiente programa:

```
1.   n: integer            --- global declaration
2.   procedure first
3.       n:=1
4.   procedure second
5.       n: integer     --- local declaration
6.       first()
7.   n:=2
8.   if read_integer() > 0
9.       second()
10. else
11.      first()
12. write_integer(n)
```

    (a) ¿Cuál es el resultado de ejecutar este programa considerando alcance estático?

    (b) ¿Cuál es el resultado de ejecutar este programa considerando alcance dinámico?

El resultado depende de las reglas de alcance y, en el caso del alcance dinámico, del valor que introduzcamos inicialmente. En el alcance estático la línea 12 imprime 1 y en el dinámico, imprime 2 si el entero introducido es positivo y 1 en caso contrario. La diferencia se debe a cuándo la línea 3 se refiere a la variable global o a la variable local declarada en la línea 5.

ALCANCE ESTÁTICO: El ámbito de una variable es el fragmento de código sintácticamente ms próximo a la declaración de la variable global `n`, que es el procedimiento `first`, el cual cambia `n` a 1 y es lo que imprime la línea 12 del programa.

ALCANCE DINÁMICO: Se elige el enlace activo más reciente en la ejecución para la variable `n`. Creamos un enlace para `n` cuando entramos en el programa principal. Creamos otro enlace para `n` cuando entramos en el procedimiento `second`. Cuando ejecutamos la sentencia de asignación de la línea 3, la `n` a la que nos estamos refiriendo dependerá de si hemos entrado por `first` o por `second`.

- si se introduce un valor > 0, llamamos a `second` con lo que cuando se llama a `first` la `n` es la local al subprograma `second` (línea 5). Este enlace se destruye al terminar la ejecución de `second` por lo que la línea 12 imprime el enlace a la variable `n` establecido por el programa principal (línea 7).
- En cualquier otro caso, llamamos a `first` directamente que modifica la variable global `n` cambiándola a 1.

16. ¿Verdadero o falso?

    (a) Si un lenguaje permite recursión, las variables locales de los subprogramas recursivos no pueden almacenarse estáticamente pero aun así pueden almacenarse en una pila. CIERTO

    (b) El recolector de basura (*garbage collector*) se encarga de la liberación automática de los recursos de memoria para un programa. CIERTO

    (c) Una pila es un tipo de almacenamiento que sigue un order "primero en entrar, primero en salir". FALSO

    (d) Un heap o montículo es una región de almacenamiento en la cual los subbloques de memoria son asignados y liberados en tiempos fijos. FALSO

17. Indica el paradigma al que pertenecen las siguientes características:

    (a) La asignación destructiva. IMPERATIVO

    (b) Las variables lógicas. LÓGICO

    (c) Orden superior. FUNCIONAL

    (d) Las instrucciones de control (while, if, for,...). IMPERATIVO

Parte II: Test

18. C
19. A
20. C
21. B
22. A
23. D
24. C
25. B
26. C
27. B
28. D
29. A
30. B
31. B
32. C
33. A