Ejercicios LTP

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
```

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
```

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
```

3. ¿Verdadero o falso?:
   (a) Una subclase es generalmente más pequeña que su superclase.
   (b) Un objeto de una subclase es también un objeto de su superclase.
   (c) Todos los métodos en una clase abstracta deben declarase abstractos.
   (d) En un programa Java no puede haber dos métodos con el mismo nombre.
   (e) Una subclase de una clase abstracta que no implementa todos los métodos abstractos que hereda debe ser abstracta.
   (f) Todos los métodos en una super clase abstracta tienen que ser abstractos.

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

//**************************************
// 2.
//**************************************
    int x1=1,y1=1;
    o.add(x1,y1);
    float x2=1,y2=1;
    o.add(x2,y2);

//**************************************
// 3.
//**************************************
    int x3=2,y3=1;
    o.subtract(x3,y3);

//**************************************
// 4.
//**************************************
    // 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.
//**************************************
    Polo m = o.new Polo();
    o.carBrand(m);
  } // END MAIN

//**************************************
// 2.
//**************************************
  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.
//**************************************
  public void subtract (float x, float y) {
    System.out.println("Double subtraction: ");
    System.out.println(x-y);
    System.out.println();
  }

//**************************************
// 4.
//**************************************
  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;
}
```

6. ¿Verdadero o falso?:

   (a) Todos los lenguajes de programación permiten la reflexión.

   (b) La reflexión puede usarse para observar y modificar el programa en tiempo de ejecución.

   (c) Los lenguajes orientados a objetos no soportan la reflexión.

   (d) La reflexión es una estrategia clave para el polimorfismo.

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);
```

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.

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

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.

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

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.

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

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.

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

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.

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

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.

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

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;          // end one

9      procedure two;
10       var x: integer;
11       begin
12         x:= 12;
13         three;
14       end;          // end two

15     procedure three;
16       begin
17         writeln(x);
18       end;          // end three

19     begin           // begin main
20       x:= 14;
21       one;
22       two;
23     end
```

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

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

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?

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.

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

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

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

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

(a) La asignación destructiva.

(b) Las variables lógicas.

(c) Orden superior.

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

Parte II: Test

18. El siguiente programa Java:

```
int y;
x = 42+y;
```

A Es correcto ya que Java tiene tipificación implícita.

B Es incorrecto: la variable `y` no ha sido declarada.

C Es incorrecto: la variable `x` no ha sido declarada.

D Es correcto gracias a la inferencia de tipos de Java.

19. Los tipos se describen mediante un lenguaje de expresiones de tipo. Indica cuál de las siguientes afirmaciones es **FALSA**:

A Los tipos básicos o primitivos no forman parte de las expresiones de tipo.

B Las variables de tipo representan tipos.

C Los constructores de tipos se usan para obtener nuevos tipos.

D Las reglas de construcción de expresiones de tipo describen cómo construir nuevas expresiones de tipo.

20. ¿Cuál es la utilidad de los tipos en un lenguaje de programación?

A Su principal utilidad es que, si el programa no tiene errores de tipos, se garantiza que no habrán errores de ejecución.

B Los tipos son esenciales para poder definir una semántica dinámica del lenguaje de programación.

C Los tipos ayudan a detectar errores de programación.

D Los tipos no sirven para nada.

21. El siguiente programa Java:

```
public class C {
  int ejemplo(int x, int y) {...}
  void ejemplo(char x) {...}
}
```

A Es un ejemplo de declaración de métodos con polimorfismo universal.

B Es un ejemplo de declaración de métodos sobrecargados (polimorfismo ad-hoc).

C Es incorrecto: la variable `x` se utiliza con tipos distintos en dos funciones distintas.

D Es un ejemplo de declaración de métodos genéricos.

22. ¿Cuál de las siguientes declaraciones define una clase genérica en Java?

A `public class foo<T> { ... }`.

B `public class foo<T k> { ... }`.

C `public <T> class foo(){ ... }`.

D `public <T> class foo { ... }`.

23. Con respecto a la siguiente definición de método genérico, indica la opción **CORRECTA**:

```
public static < E > void printArray( E[] inputArray )
{    ...   }
```

A La variable genérica `E` debe haber sido declarada en la clase que contiene a este método genérico. Por ejemplo así:
```
class Generica {
   Figura E;
   public static < E > void printArray( E[] inputArray ){...}
   ...
}
```

B La variable genérica `E` debe haber sido declarada en la llamada a este método genérico. Por ejemplo así: `printArray(Figura E)`.

C Este método genérico debe pertenecer a una clase genérica o a una clase que herede de una clase genérica.

D Aunque sea genérico, este método puede pertenecer a una clase no genérica.

24. Dadas las siguientes definiciones de clases

```
public class Animal {
}
public class Dog extends Animal {
}
```

Indica cuál de las siguientes asignaciones es **INCORRECTA**:

A
```
Animal animal1 = new Dog();
```

B
```
Animal animal1 = new Dog();
Dog animal2 = (Dog) animal1;
```

C
```
Dog animal1 = new Animal();
```

D
```
Animal animal1 = new Dog();
Animal animal2 = animal1;
```

25. Indica cuál de las siguientes afirmaciones sobre la reflexión es **FALSA**:

A Se trata de una característica que permite inspeccionar y/o modificar la ejecución de un programa en tiempo de ejecución.

B Todos los lenguajes de programación tienen alguna biblioteca o conjunto de funciones que proporcionan reflexión.

C Resulta útil en tareas de metaprogramación.

D Si se usa mal, puede afectar al rendimiento y seguridad del programa objeto.

26. En un lenguaje de programación imperativo:

A No hay efectos laterales.

B No hay tipos.

C El programa es una secuencia de instrucciones que puede cambiar su estado.

D El programa consiste en una descripción lógica del problema a resolver.

27. Indica cuál de las siguientes afirmaciones es **CIERTA**:

A La lógica de un programa declarativo se expresa usando instrucciones de control.

B Un programa declarativo puede entenderse como una especificación ejecutable de un problema.

C La programación en el paradigma imperativo se basa en la definición de ecuaciones.

D En el paradigma lógico y en el paradigma funcional se dan efectos laterales tal y como ocurre en el paradigma imperativo.

28. Indica cuál de las siguientes afirmaciones sobre paradigmas es **FALSA**:

A En los paradigmas basados en interacción, un programa es una comunidad de entidades que interactúan siguiendo ciertas reglas de interacción.

B La abstracción, el encapsulamiento, la modularidad y la jerarquía son elementos fundamentales del paradigma orientado a objetos.

C En el paradigma de programación dirigida por eventos, el flujo del programa está determinado por eventos o mensajes.

D El objetivo de la programación paralela consiste en controlar el flujo del programa mediante eventos.

29. Indica cuál de las siguientes afirmaciones referentes a la Programación Declarativa (PD) e Imperativa (PI) es **FALSA**:

A Las instrucciones de un programa declarativo son un conjunto de inferencias lógicas.

B Un programa en PD se corresponde con la especificación de un problema.

C El modelo de computación de la PI es una máquina de estados.

D Las variables en la PI representan referencias a la memoria.

30. Indica cuál de las siguientes afirmaciones es **CIERTA**:

A una desventaja de los lenguajes de programación declarativos es que, para resolver un mismo problema, se requieren en general más líneas de código que usando otros lenguajes de programación más convencionales, como Pascal.

B al ser de mayor nivel, los programas declarativos resultan más fáciles de mantener que los correspondientes programas imperativos.

C no existe ningún lenguaje orientado a objetos y a la vez declarativo.

D los lenguajes declarativos carecen de aplicaciones prácticas.

31. Indica cuál de las siguientes asociaciones entre lenguaje de programación y concepto (o propiedad) inherente al mismo es **CIERTA**:

A lenguaje imperativo - herencia

B lenguaje funcional - polimorfismo

C lenguaje lógico - estados explícitos

D lenguaje imperativo - orden superior

32. Indica cuál de las siguientes asociaciones es **FALSA**:

A Paradigma imperativo $\Leftrightarrow$ la ejecución del programa consiste en construir los estados de la máquina necesarios para llegar a la solución.

B Paradigma declarativo $\Leftrightarrow$ las instrucciones son fórmulas de algún sistema lógico.

C Paradigma imperativo $\Leftrightarrow$ se satisface el principio "PROGRAMA = ESPECIFICACION DE UN PROBLEMA".

D Paradigma declarativo $\Leftrightarrow$ se satisface el principio "PROGRAMA = LOGICA + CONTROL".

33. Indica cuál de las siguientes afirmaciones es **CIERTA**:

A El inicio de la Programación Concurrente está en la invención de la interrupción a finales de los 50.

B En la programación imperativa se reduce el gap semántico entre especificación y programación.

C En el paradigma declarativo el concepto básico es el estado, que se define por los valores de las variables involucradas en cada punto de la computación.

D Java es un lenguaje de programación sin herencia.