COMPLEMENTOS DE PROGRAMACIÓN

EXAMEN SEPTIEMBRE 2013

1. **Indica lo que escribe el programa dado en la salida estándar, explicando por qué es así.**

El resultado del programa será:

Persona

Empleado

Profesor

En el programa principal se declara una variable profesor1 de tipo Profesor que en este caso no hace nada. Después se crea un objeto de tipo Profesor llamado profesor2 que, al inicializarlo, se pasa a realizar el constructor que tiene que realizar. En este caso como la clase Profesor extiende de Empleado y esta clase a su vez de Persona, al entrar en el constructor de Profesor(), llevará al constructor de Empleado() y por último llevará al constructor de Persona() donde se ejecutará el System.out.prinln("Persona"). Tras esto se volverá al constructor de Empleado y se ejecutarán las sentencias que se encuentre dentro de él tras lo cual volverá al constructor de Profesor don por último se ejecutarán las sentencias que haya dentro del dicho constructor.

2. **Indicar cuál es la salida de los programas dados y explicar por qué es así.**
    a) **Programa 1.**

    El resultado del programa será:

    10.0

    10.0

    Como el objeto creado es de la clase A, aunque extienda de la clase B, tiene un método sobrescrito de la clase B. Por tanto cuando se llama a dicho método pasándole tanto un int como un double, ejecutará el método sobrescrito de la clase A.

    b) **Programa 2.**

    El resultado del programa será:

    10

    10.0

    El método p de la clase A pasa como parámetro un int, por tanto cuando se llama primero al método p desde el objeto a con un entero, se ejecutará el método de la clase A.

Sin embargo el método de la clase B pasa como parámetro un valor doublé, luego en el caso en el que se llama al método p con el double 10.0, al comprobar que el método sobrescrito en la clase A solo acepta como parámetro valores int, se ejecutará el método de la clase B que si acepta parámetros dobule.

3. **Indica cuales son las diferencias entre las excepciones comprobadas y las no comprobadas en Java.**

Las excepciones comprobadas hacen que el compilador obligue a tratarlas con un bloque try-catch o bien ser declaradas en la cabecera del método. Las excepciones de la clase Exception y sus subclases son comprobadas.

Sin embargo las excepciones no comprobadas en la mayoría de los casos son errores lógicos que deben arreglarse modificando el programa y Java no obliga a capturar estas excepciones. Las excepciones que son representadas por las clases RunTimeException y Error son no comprobadas.

4. **(Este año 2015 no hemos llegado a ver tan en profundidad las pilas por lo que no cae).**
5. **Usa el framework Fork/Join para construir un programa paralelo que encuentre el valor máximo de un array de int.**

Usar el framework Fork/Join implica que dividamos el problema que se nos propone en muchos sobproblemas para su tratamiento (Fork). Cuando todos estos subproblemas acaban se unen de nuevo para mostrar la solución (Join). Para construir el programa que se nos propone dividiremos el array dado en dos mitades, buscamos el máximo de cada mitad, y devolvemos el máximo entre los dos valores encontrados.

El máximo de cada mitad se busca recursivamente de la misma manera hasta que se llegue a un array con un tamaño por debajo de un determinado umbral, a partir de aquí usaremos un algoritmo secuencial.

El programa quedaría tal que:

```
//En primer lugar importamos el paquete de java concurrent
import java.util.Scanner;
import java.util.concurrent.*;

//Implementamos la clase que contiene al main
public class MaximoParalelo {
   private static final Scanner input = new Scanner(System.in);
   private static int dimension;

   public static void main(String[] args){
      //Creamos la dimension del array de enteros
      System.out.println("Introduzca la dimension del array de enteros: ");
      dimension = input.nextInt();
      //Creamos el array de enteros
      int[] arrayEnteros = new int[dimension];
      //Introducimos los valores en el array
      for(int i = 0; i < arrayEnteros.length; i++){
         arrayEnteros[i] = i;
      }

      System.out.println("El número máximo del array es: " + max(arrayEnteros));
   }

   private static class MaximoTask extends RecursiveTask<Integer>{
      //Especificamos el umbral
      private final static int umbral = 100;
      //Creamos el array de enteros y sus posiciones inferior y superior
      private int[] arrayEnteros;
```

```
    private int posInferior;
    private int posSuperior;

    //Creamos el constructor de MaximoTask
    public MaximoTask(int[] array, int inferior, int superior){
       this.arrayEnteros = array;
       this.posInferior = inferior;
       this.posSuperior = superior;
    }

    //Este métododefine como se realiza la tarea y lo sobrescribimos para
    //implemetar el algoritmo recursivo
    @Override
    public Integer compute(){
       if(posSuperior - posInferior < umbral){
          //Algoritmo para hayar el maximo valor del array
          int maximo = arrayEnteros[0];
          for(int i = posInferior; i < posSuperior; i++){
             if(arrayEnteros[i] > maximo){
                maximo = arrayEnteros[i];
             }
          }
          return maximo;
       }
       else{
          //Hallamos la posicion media del array para hacer la division
          int posicionMedia = (posInferior + posSuperior) / 2;
          //Utilizamos RecursiveTask para dividir el vector
          RecursiveTask<Integer> izquierda = new MaximoTask(arrayEnteros,
posInferior, posicionMedia);
```

```
            RecursiveTask<Integer> derecha = new MaximoTask(arrayEnteros,
posicionMedia, posSuperior);
            //Colocamos las tareas derecha e izquierda para ser ejecutadas
            //con Fork() de manera asincrona
            derecha.fork();
            izquierda.fork();
            //Devolvemos el maximo valor de ambos arrays cuando acaben todas
            //las tareas, indicado mediante join()
            return Math.max(izquierda.join(), derecha.join());
        }
    }
  }

  //Esta clase sirve para paralelizar el algoritmo de encontrar el maximo
  //mediante el uso RecursiveTask porque nos devuelve un valor
  public static int max(int[] array){
    RecursiveTask<Integer> task = new MaximoTask(array, 0, array.length);
    //Creamos un ForkJoinPool con todos los procesadores disponibles
    ForkJoinPool pool = new ForkJoinPool();
    //Devolvemos el resultado de la tarea cuando acabe mediante el metodo
    //invoke
    return pool.invoke(task);
  }
}
```