Departamento de
Informática e Ingeniería
de Sistemas
Universidad Zaragoza
1542

Examen de Teoría

5 de julio de 2023

Tecnología de Programación
Grado en Ingeniería Informática
Duración: 2 horas 30 minutos

Recuerda:

La evaluación final consta de las siguientes pruebas:

Prueba escrita → 60 % de la nota (mínimo 5 para aprobar)
Prácticas y Problemas → 40 % de la nota

El examen se realiza de forma **INDIVIDUAL**.

Entrega:

- Escribe tu nombre completo en **todas** las hojas del examen.
- Responde a cada ejercicio comenzando **en una hoja diferente**, se entregan por separado.
- No se puede entregar la resolución escrita a **lápiz**.

Ejercicio 1 **[2 puntos]**

Dentro de la Programación Orientada a Objetos, el establecimiento de una relación de **herencia** entre dos clases desencadena una serie de mecanismos que permiten **polimorfismo por inclusión**.
Sobre este tema realiza las siguientes tareas:

1. En el lenguaje orientado a objetos de tu elección (C++ o Java) escribe un **pequeño ejemplo de código** que incluya:
   - Dos clases no abstractas relacionadas mediante herencia que permitan polimorfismo por inclusión, estableciendo comportamientos diferentes de un mismo método.
   - Una función (o método estático de otra clase) no genérico que permita como parámetro las dos clases anteriores y que invoque al menos a un método que pueda tener diferentes comportamientos dependiendo de cuál de las dos clases sea.
   - Un programa principal que invoque a la función correspondiente con objetos pertenecientes a las dos clases.
   - Asegúrate que el diferente comportamiento de ambas dos clases se ve reflejado, de alguna forma, en la salida estándar: que los métodos, la función o el programa principal muestren algo por pantalla que dependa del comportamiento particular de cada clase.

2. **Explica** dónde se ve en tu ejemplo anterior el polimorfismo por inclusión y qué mecanismos de la herencia lo están permitiendo.

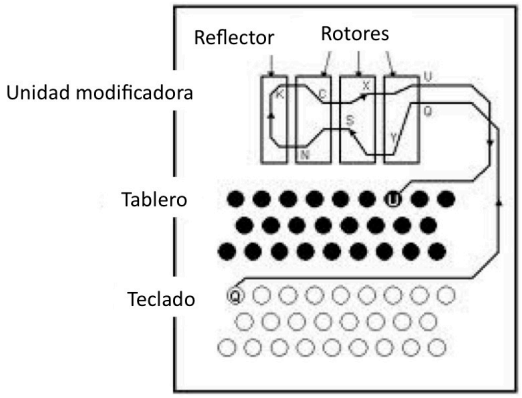
3. Explica qué tipo de asociación de métodos (estática o dinámica) está siendo aplicado en tu ejemplo, y por qué. Escribe dos **trazas de ejecución** (lo que muestra el programa por pantalla) de tu programa principal:
   - Una traza suponiendo que la asociación de métodos es siempre estática.
   - Una traza suponiendo que la asociación de métodos es siempre dinámica.

Ejercicio 2 **[4 puntos]**

La **Máquina Enigma** fue un dispositivo electromecánico usado en la Segunda Guerra Mundial por el ejército alemán para cifrar sus comunicaciones secretas, principalmente para la transmisión de objetivos a su flota de submarinos, de forma que los aliados no pudieran interceptarlas.

Está compuesta por varios discos (rotores) que implementan un cifrado por sustitución de unos caracteres por otros, directamente mediante cables. Al pulsar una letra en el teclado, se iluminaba la lámpara correspondiente al carácter codificado.

(a) Máquina Enigma y rotor



(b) Esquema

Cada rotor implementa físicamente una sustitución de caracteres, que se puede representar mediante una correspondencia entre dos cadenas de caracteres:

```
"ABCDEFGHIJKLMNOPQRSTUVWXYZ" -> "EKMFLGDQVZNTOWYHXUSPAIBRCJ"
```

Para simplificar, sólo se trabaja con letras mayúsculas, y el resto de caracteres quedan sin cifrar. Las máquinas Enigma más sencillas sólo disponían de 3 rotores, pero luego fueron apareciendo versiones con 4 o más discos. Al aplicar varias sustituciones en secuencia, el cifrado es prácticamente indescifrable. Hasta que Alan Turing, uno de los padres de la Informática, comenzó el desarrollo de un computador y de algoritmos que pudieran descifrarlo... pero eso es otra historia.

Se pide que definas/implementes en Haskell lo siguiente:

- Un tipo de datos **`Enigma`** que represente la máquina de cifrado, con un número indefinido de **rotores** (cadenas de sustitución).
- Una función

  ```
  encode :: Enigma -> Char -> Char
  ```

  que implemente el cifrado de un carácter, aplicando en secuencia el cifrado definido por cada uno de los rotores.
- Una función

  ```
  enigma :: Enigma -> String -> String
  ```

  que implemente el cifrado de un texto.

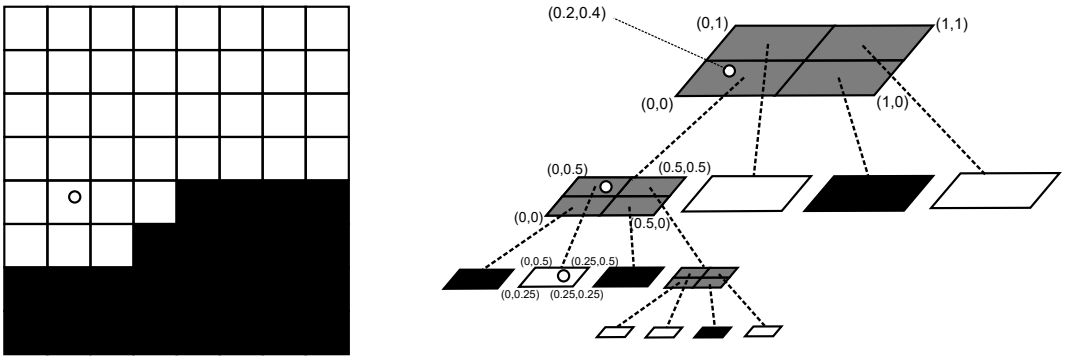
Se valorará la concisión, brevedad y eficiencia de la solución propuesta.
Se permite utilizar las funciones de la biblioteca estándar de Haskell que consideres necesarias.
Se permite definir nuevas funciones Haskell, pero recuerda que se tendrá en cuenta la eficiencia de la solución, que podría verse afectada por las funciones que se reutilicen entre apartados.

Ejercicio 3 **[4 puntos]**

Para el desarrollo de un videojuego en dos dimensiones y la detección de colisiones, se necesita almacenar la información de ocupación del mapa (si un punto del mapa está o no ocupado). Habitualmente, esto podría hacerse mediante una imagen en blanco y negro, que representa una matriz bidimensional de valores lógicos (booleanos) donde *verdadero* (blanco) significa zona ocupada, y *falso* (negro), zona libre. Sin embargo, los mapas en este videojuego son masivos (muy grandes) y por tanto imposibles de guardar en memoria, así que se pretende guardar esta información en un ***quadtree***.

Un *quadtree* (árbol cuádruple) es una estructura de datos jerárquica utilizada para organizar información espacial en dos dimensiones. El término *quadtree* se deriva de *quad* (cuádruple) y *tree* (árbol), ya que el árbol se divide en cuatro hijos en cada nivel. El espacio se divide en cuadrantes más pequeños de manera recursiva hasta que se alcanza cierto nivel de detalle. El árbol tiene dos tipos de nodos:

- Los **nodos hoja** representan una zona del espacio indivisible, que puede estar ocupada o no ocupada.
- Los **nodos internos** representan una zona del espacio dividida en cuatro cuadrantes, cada uno de ellos siendo un nodo del árbol.



Izquierda: *la imagen en blanco y negro (de* 8×8 *píxeles) que representa la ocupación de un mapa.* ***Derecha:*** *quadtree que representa dicha imagen. El círculo blanco representa un punto para el que se quiere comprobar su ocupación, en la posición* $(0{,}2, 0{,}4)$*. Para comprobar la ocupación en los nodos internos, hay que buscar a qué cuadrante corresponde el punto y recursivamente explorar ese nodo. Eventualmente, llegamos a un nodo hoja que nos indica que está ocupado.*

Se pide desarrollar, en un lenguaje orientado a objetos (C++ o Java), las clases necesarias para representar un *quadtree*. Tu representación debe ocupar la mínima cantidad de memoria posible, es decir, un nodo hoja no deberá almacenar ningún tipo de información de sus hijos (porque no tiene) mientras que un nodo interno no deberá almacenar información sobre su ocupación (porque esa información está en sus hijos).

Adicionalmente, y asumiendo que el *quadtree* completo (su nodo raíz) representa un espacio indexado por coordenadas reales entre 0 y 1 tanto para la dimensión horizontal como vertical, implementa un método

```
ocupado(float x, float y)
```

que devuelva si el punto x, y está ocupado o no ocupado en el *quadtree*. Para ello, en los nodos internos, deberá elegir cual de los cuatro cuadrantes explorar (dependiendo de si x e y corresponden a menos o más de la mitad del espacio ocupado por el nodo) y explorarlo recursivamente, mientras que los nodos hoja directamente devuelven el estado de ocupación. Valores de x e y fuera del rango entre 0 y 1 se considerarán ocupados.

Nótese que **no se pide** construir el árbol, solo recorrerlo, asumiendo que está construido correctamente (esto implica que no es necesario que gestiones la memoria). La solución deberá ser escalable, de tal forma que añadir un nuevo tipo de nodo se pueda hacer con facilidad sin necesidad de retocar todo el código de las clases o métodos existentes. Puedes utilizar todas las clases y herramientas definidas en la biblioteca estándar del lenguaje elegido.