Facultad de Informática - Universidad Complutense de Madrid

Tecnología de la Programación I - Curso 2023/2024

Examen Ordinario - Parte teórica

Apellidos, Nombre: _____

Instrucciones

- Duración máxima: 1 hora
- Puntuación máxima: 3 puntos
- Cuando entregues este examen teórico, se te proporcionará el enunciado del examen práctico

1. **[1 punto]** Dadas las siguientes clases e Interfaces:

   a) Indica los errores de compilación del siguiente código y el motivo.

   b) Indica los cambios que tendrías que hacer para que el programa mostrara la siguiente salida, usando solamente strings devueltos por el método tocar para construir la salida del programa y usando un instrumento para cada línea de la salida.

   ```
   ba
   DUM
   tss
   do (flauta)
   do (flauta)
   do (flauta)
   re (flauta)
   mi (flauta)
   doooooooooo
   ```

```java
abstract class Instrumento {
  public abstract String tocar();
}

class Percusion extends  Instrumento {

   private String onomatopeya;

   public Percusion(String string) {
      this.onomatopeya = string;
   }

   public Percusion() {
      this("Pum!");
   }

   @Override
   public String tocar() {
      return onomatopeya;
   }
}

interface TocaNotas {
   public abstract String tocar (String nota) ;
}
```

```
27 class Flauta extends Instrumento  implements TocaNotas {

29    private String notas [] = {"do", "re", "mi", "fa", "sol", "la", "si"};

31    @Override
32    public String tocar() {
33       return this.tocar("do");
34    }

36    private boolean notaCorrecta(String nota) {
37       for (String n: notas) {
38          if (n.equals(nota)) return true;
39       }
40       return false;
41    }

43    @Override
44    public String tocar(String nota) {
45       String sonido = "shh";
46       if (notaCorrecta(nota)) {
47          sonido = nota;
48       }
49       return sonido + " (flauta)";
50    }
51 }

53 public class Main {
54
55    public static void main(String[] args) {
56       Instrumento caja = new Percusion("ba");
57       Instrumento bombo = new Percusion();
58       Instrumento plato = new Percusion("tss");
59       Flauta flauta = new Flauta();
60       TocaNotas flautaDulce = new Flauta();
61       Instrumento flautaDulce = new Flauta();
62
63         Instrumento[] orquesta;
64       orquesta[0] = caja;
65       orquesta[1] = bombo;
66       orquesta[2] = plato;
67       orquesta[3] = flauta;
68       orquesta[4] = flautaDulce;

70       bombo = new Percusion("DUM");

72       for (Instrumento instrumento : orquesta) {
73          System.out.println(instrumento.tocar());
74       }

76       String sonidos = {
77             orquesta[3].tocar("do"),
78             flautaDulce.tocar("re"),
79             flauta.tocar("mi mi mi"),
80             orquesta[4].tocar("doooooooooo")
81       };

83       for (String sonido : sonidos) {
84          System.out.println(sonido);
85       }
86    }
87 }
```

2. **[1 punto]** Dado el código presentado a continuación, contesta razonadamente a las preguntas posteriores.

```
1  public class PosicionIncorrecta extends Exception {
2     public PosicionIncorrecta(String message) {
3        super(message);
4     }
5  }

7  public abstract class Ficha {
8     protected boolean esBlanca = true;
9     protected int posX = 0;
10    protected int posY = 0;
11    protected boolean viva = true;

13    public Ficha(boolean blanca, int x, int y) {
14       this.esBlanca = blanca;
15       this.posX = x;
16       this.posY = y;
17    }

19    public boolean come(Ficha otra, int x, int y) throws PosicionIncorrecta {
20       boolean haComido = false;
21       // Si la otra pieza está viva
22       if ( otra.viva) {
23         // Y está en la misma posición
24         if (x == otra.posX && y == otra.posY) {
25             // y es de distinto color
26             if (this.esBlanca != otra.esBlanca) {
27                // mata a la otra ficha
28                otra.viva = false;
29                // y se mueve a la posición
30                posX = x;
31                posY = y;
32                haComido =true;
33             } else {
34                throw new PosicionIncorrecta("¡No puedes comer una pieza del
     mismo color!");
35             }
36          }
37       }
38       return haComido;
39    }

41    public  void mueve(int x, int  y) throws PosicionIncorrecta {
42       if (!this.viva) throw new PosicionIncorrecta("no puedes mover una pieza
     muerta");
43       if (x < 0 || x > 7) throw new PosicionIncorrecta("columna incorrecta");
44       if (y < 0 || y > 7) throw new PosicionIncorrecta("fila incorrecta");
45       this.posX = x;
46       this.posY = y;
47    };
48 }

50 public class Peon extends Ficha {

52    public Peon(boolean blanca, int x, int y) {
53       super(blanca, x, y);
54    }

56    @Override public boolean come(Ficha otra, int x, int y) throws
      PosicionIncorrecta {
```

```
57          if (this.movimientoComerValido(x,y)) {
58              if (super.come(otra, x, y)) {
59                  return true;
60              }
61          }
62          return false;
63      }

65      @Override
66      public void mueve(int x, int y) throws PosicionIncorrecta{
67          if (this.movimientoSimpleValido(x, y)) {
68              try {
69                  super.mueve(x,y);
70              }
71               catch (PosicionIncorrecta e){
72                  System.out.println(
73                          "Excepcion capturada al intentar mover una pieza: " + e.
    getMessage());
74              }
75          } else {
76              throw new PosicionIncorrecta(
77                  "los peones sólo pueden moverse una posición y hacia delante");
78          }
79      }
80      public boolean movimientoSimpleValido(int x, int y) { // ... }
81      public boolean movimientoComerValido(int x, int  y) { // ... }
82  }

84  public class Main {
85      static Peon blanco1 = new Peon(true, 1, 1);
86      static Peon blanco2 = new Peon(true, 2, 2);
87      static Peon negro1 = new Peon(false, 0, 0);
88      static Peon negro2 = new Peon(false, 3, 3);

90      static Ficha piezas [] = {blanco1, blanco2, negro1, negro2};

92      private static void coloca(Ficha f, int x, int y) throws PosicionIncorrecta {
93          boolean haComido = false;
94          try {
95              for (Ficha p: piezas) {
96                  if (f.come(p, x, y)) {
97                      haComido = true;
98                  }
99              }
100         }
101         catch (PosicionIncorrecta e) {
102             System.out.println("Excepción capturada al intentar comer una pieza:" +
     e.getMessage());
103             return;
104          }
105         if(!haComido) {
106             f.mueve(x, y);
107         }
108     }

110     public static void main(String[] args) {
111         try {
112             coloca(blanco1, 2, 2);
113             coloca(blanco2, 3, 3);
114             coloca(negro2, 3, 2);
115             coloca(negro1, 0,-1);
```

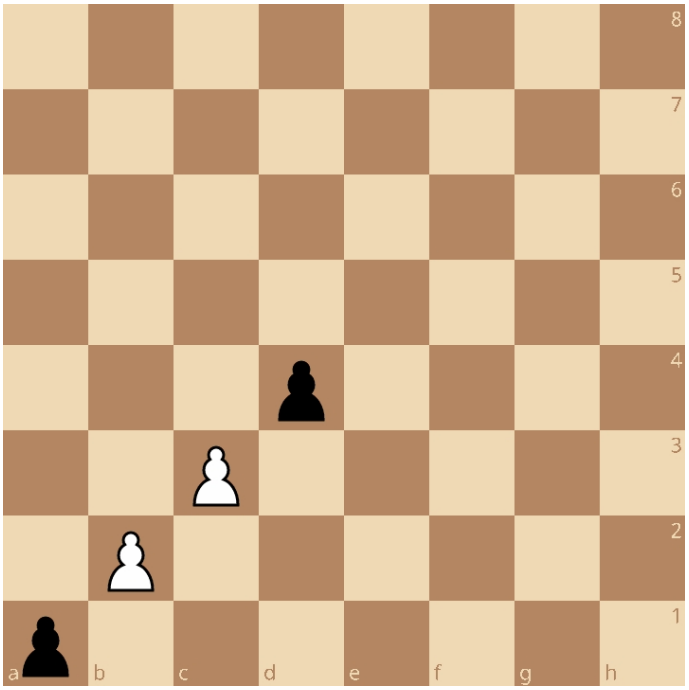

Figura 1: Posición inicial del tablero del ejercicio 2.

```
117         } catch (PosicionIncorrecta e) {
118             System.out.println("Excepción capturada en Main: " + e.getMessage());
119         }
120     }
121 }
```

a) Indica la salida del programa.

Nota 1: El programa implementa una versión simplificada del ajedrez, donde solo existen peones que se mueven una posición hacia delante, o comen moviéndose en cualquier diagonal una posición hacia delante. Así mismo, moverse hacia delante para las piezas blancas significa aumentar su posición `posY`, y para las negras, reducirla. No están implementados otras características como el manejo del turno, u otros movimientos del peón (como el poder avanzar dos casillas, o la captura al paso). Asume que:

- El método `movimientoSimpleValido` devuelve `true` siempre que el parámetro `y` esté una posición por delante de la posición actual de la ficha y que el parámetro `x` y la posición x de la ficha sean iguales y devuelve `false` en caso contrario.
- El método `movimientoComerValido` devuelve `true` si el parámetro `y` está una posición por delante de la posición actual de la ficha y el parámetro `x` una posición hacia la derecha o hacia la izquierda y `false` en caso contrario.
- Estos dos métodos no comprueban si la posición es válida en el tablero. Por ejemplo, un peón blanco colocado en la posición (0,7) devolvería true al ejecutar su método movimientoSimpleValido(0,8).

Nota 2: Las posiciones se representan con enteros del 0 al 7. Así, la posición A2 se representa con las coordenadas x=0 y=1.

Pista: Hay tres movimientos `coloca(...)` inválidos en el main por alguna razón en la que el código lanzaría una excepción en caso de ejecutarse individualmente. El otro movimiento es válido y no lanzaría excepción.

3. **[1 punto]** Dado el código presentado a continuación, contesta razonadamente a las preguntas posteriores.

```
1  public interface Acelera {
2     public abstract void acelera() throws VelocidadException;
3     public abstract int getVelocidad();
4  }

6  public class Bici {
7     public int velocidad = 0;
8  }

10 public class Coche {
11    public int velocidad = 0;
12 }

14 package examen.ejercicio3;

16 public class Main {
17    public static void main(String[] args) {
18      Object vehiculos [];
19       vehiculos[0] = new Bici();
20       vehiculos[1] = new Coche();
21       boolean terminar = false;

23       while (!terminar) {
24          for (Object o: vehiculos) {
25             if (o instanceOf Coche) {
26                o.velocidad = o.velocidad + 10;
27                if (o.velocidad > 120) {
28                   System.out.println("Vehiculo multado por exceso de velocidad")
    ;
29                   terminar = true;
30                }
31             }
32             else if (o instanceOf Bici) {
33                o.velocidad = o.velocidad + 2;
34                if (o.velocidad > 40) {
35                   System.out.println("Ciclista exhausto");
36                   terminar = true;
37                }
38             }
39          System.out.println("Velocidad: " o.velocidad);
40          }
41       }
42    }
43 }
```

a) Dada la interfaz `Acelera`, indica los cambios que habría que hacer en las clases `Coche` y `Bici` para que pudieran utilizarla siendo ambos tipos de objetos que pueden acelerar. Indica también si hay cambios relativos a sus atributos.

b) Proporciona la implementación del método acelera() de Bici de forma acelere de 2 en 2 y que lance una excepción `AceleraException` con el mensaje *Çiclista Exhausto"* cuando supere la velocidad de 40.

c) Indica los cambios que habría que hacer en el bucle principal y la inicialización de las variables del método main para poder tratar las bicis y los coches como objetos que aceleran y no hacer distinción de casos según la clase de cada instancia en el método main. Proporciona la implementación del bucle principal. Puedes suponer que el método acelerar de Coche está implementado de manera análoga a tu respuesta para el apartado b. Puedes usar tu conocimiento del comportamiento del programa, que acelerará ambos objetos hasta que uno lance una excepción para

diseñar el flujo de tu programa.

Nota 1: La salida del programa deberá ser la siguiente (entendiendo los puntos suspensivos como continuación de la serie de salidas de velocidad).

```
Velocidad: 2
Velocidad: 10
Velocidad: 4
Velocidad: 20
...
Velocidad: 26
Vehículo multado por exceso de velocidad
Velocidad:130
```

Nota 2: Solo se pueden imprimir mensajes de las siguientes formas:

- Tras capturar una excepción haciendo uso del mensaje de la excepción.
- Una vez por cada elemento de la lista vehículos en cada iteración del bucle, haciendo uso del método `getVelocidad`.