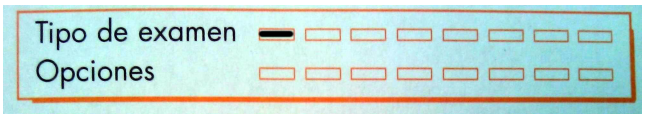
Sólo existe una respuesta acertada. Los errores restan un tercio de lo que suman los aciertos.



Conceptos avanzados: excepciones, paquetes y documentación

1. Una excepción
   - **a. es una situación anómala que provoca que se interrumpa la ejecución (aunque posteriormente pueda continuar si es adecuadamente capturada)**
   - b. es una situación anómala que se notifica hacia arriba pero sin interrumpir la ejecución normal
   - c. si se captura adecuadamente no interrumpe la ejecución ni siquiera parcialmente
   - d. ninguna de las anteriores
2. Sobre `throw` y `throws`
   - a. ambas son declarativas
   - b. ambas se usan dentro del cuerpo de los métodos para lanzar excepciones
   - c. `throw` es declarativa y `throws` se usa dentro del cuerpo de los métodos para lanzar excepciones
   - **d. `throws` es declarativa y `throw` se usa dentro del cuerpo de los métodos para lanzar excepciones**
3. ¿Quién puede lanzar excepciones?
   - **a. cualquier método o constructor**
   - b. sólo los métodos, los constructores no pueden lanzar excepciones
   - c. cualquier método y el constructor por defecto (el resto de constructores no están autorizados para lanzar excepciones)
   - d. sólo los constructores
4. Cuando hablamos de gestionar excepciones nos referimos a
   - a. propagarlas únicamente
   - b. capturarlas únicamente
   - **c. propagarlas o capturarlas**
   - d. definirlas
5. Una excepción que hereda de una subclase de `RuntimeExcepcion`
   - a. es comprobada ya que para ser no comprobada debería heredar directamente de `RuntimeException`
   - **b. es no comprobada**
   - c. es comprobada ya que `RuntimeExcepcion` hereda de `Exception`
   - d. sólo sería comprobada si se declara como tal al definirla
6. Cual de las siguientes afirmaciones es FALSA
   - **a. todas las excepciones que se lancen dentro de un bloque `try` deben poder ser capturadas por un `catch`**
   - b. las excepciones no comprobadas se pueden capturar con un `catch` igual que las comprobadas

c. en un bloque `try` pueden existir varios `catch` pero un único `finally`
d. en un bloque `try` se pueden lanzar excepciones definidas por nosotros y excepciones estándar de `java`

7. El código dentro del `finally` se ejecuta
   a. si no se ha producido ninguna excepción dentro del try
   b. si no se ha producido ninguna excepción dentro del try o la que se haya producido ha sido capturada
   c. siempre
   d. si no se ha producido ninguna excepción dentro del try o la que se haya producido es de tipo comprobada
8. Sobre las excepciones
   a. es obligatorio gestionar todas las excepciones
   b. no es obligatorio gestionar ningún tipo de excepciones
   c. sólo es obligatorio gestionar las excepciones "comprobadas"
   d. sólo es obligatorio gestionar las excepciones "no comprobadas"
9. Cuando importamos un paquete
   a. podemos usar todas las clases, interfaces y excepciones incluidas en el mismo
   b. podemos usar todas las clases del paquete, pero no los interfaces ni las excepciones
   c. podemos usar todas las clases, interfaces y excepciones incluidos en el paquete que sean públicos
   d. podremos usar únicamente los interfaces definidos como públicos dentro del paquete
10. La herramienta `javadoc` para la generación de documentación
    a. permite generar documentación del mismo estilo que la utilizada para documentar el propio API de java
    b. permite generar documentación más sencilla que la utilizada en propio API de java
    c. requiere que se incluya tags especiales para funcionar mínimamente
    d. es una herramienta para el control de excepciones

Entrada/salida

11. La clase `Scanner` se usa
    a. únicamente para leer ficheros tipo texto
    b. tanto para leer ficheros tipo texto como para leer de la entrada estándar
    c. únicamente para leer de la entrada estándar
    d. se usa tanto para ficheros de formato texto como para binarios
12. El siguiente código:

```
FileReader reader = new FileReader("input.txt");
Scanner in = new Scanner(reader);
double n = in.nextDouble();
```

    a. lee del fichero `input.txt` un `double` y se lo asigna a n
    b. da error porque abre el fichero en modo escritura

c. `Scanner` es únicamente para entrada desde consola
d. se produce un error de *casting* en la última línea

13. Si el fichero `input.txt` contiene el texto "`programación orientada a objetos`", ¿qué valor tendrá la variable `s` al final del siguiente código?

```
FileReader reader = new FileReader("input.txt");
Scanner in = new Scanner(reader);
String s = in.next();
```

a. "programación orientada a objetos"
b. -1
c. "programación"
d. 'p'

14. Si el fichero `input.txt` contiene en la primera línea un `double` que corresponde a la nota de un alumno y en la segunda línea un `String` que corresponde con el NIA del mismo, ¿qué valor tendrá la variable `s` al final del siguiente código?

```
FileReader reader = new FileReader("input.txt");
Scanner in = new Scanner(reader);
int v = in.nextInt();
String s = in.nextLine();
```

a. una cadena vacía, ya que después del `nextInt` es necesario provocar un salto de línea con `nextLine`
b. el contenido de la segunda línea del fichero, o sea, el NIA del alumno
c. el contenido de la primera línea, esto es, la nota
d. se produce una excepción cuando se intenta leer un entero

15. Para leer un fichero de tipo binario (por ejemplo un .doc de Microsoft Word), ¿usamos la clase `Reader` o `InputStream`?
a. usamos la clase `Reader`, la clase `InputStream` no se puede usar
b. usamos la clase `InputStream`, la clase `Reader` no se puede usar
c. cualquiera de las dos clases
d. ninguna de las dos clases

16. ¿Qué devuelven los métodos `read()` de las clases `Reader` y `InputStream` y por qué?
a. el de la clase `Reader` devuelve un `char` ya que se usa con ficheros de tipo texto y el de la clase `InputStream` devuelve un `byte` ya que se usa con ficheros de tipo binario
b. ambos devuelven `int` y -1 cuando se llega a fin de fichero
c. ambos devuelven un byte ya que lo que se almacena en los ficheros son bytes
d. ninguno de los anteriores

17. En acceso directo, si almacenamos en el fichero los atributos de un objeto, y contamos con el método `size()` implementado como `return (int) (file.length() / RECORD_SIZE)` lo que devolverá este método es:
a. el número de registros que hay en el fichero
b. uno menos del número de registros contenidos en el fichero
c. el número de bytes que hay en el fichero

d. la posición del puntero en el fichero

18. Para acceder al elemento que está en quinta posición de un fichero de acceso directo que contiene registros de tamaño 20, debemos posicionar en:
    a. el byte 5
    b. el byte 80
    c. el byte 100
    d. el byte 25

19. En acceso directo, la sentencia `readInt`:
    a. lee los siguientes 4 bytes del fichero que devuelve como un entero y avanza el puntero de lectura de tal forma que el siguiente `readInt` lee los siguientes 4 bytes
    b. lee los siguientes 4 bytes del fichero que devuelve como un entero y NO avanza el puntero de lectura de tal forma que el siguiente `readInt` lee los mismos 4 bytes
    c. lee los siguientes 8 bytes del fichero que devuelve como un entero y avanza el puntero de lectura de tal forma que el siguiente `readInt` lee los siguientes 8 bytes
    d. lee los siguientes 8 bytes del fichero que devuelve como un entero y NO avanza el puntero de lectura de tal forma que el siguiente `readInt` lee los mismos 8 bytes

20. Los `ObjectStreams` permiten
    a. almacenar objetos en ficheros binarios simplemente haciendo que la clase implemente el interface `Serializable`
    b. almacenar cualquier objeto en ficheros binarios
    c. almacenar objetos de clases que implementen el interface `Serializable` en ficheros de texto
    d. ninguna de las anteriores

Algoritmos de ordenación con OO

21. ¿Cuál de la principal ventaja de HeapSort en comparación con QuickSort?
    a. Heap Sort siempre es más rápido que Quick Sort
    b. El algoritmo Heap Sort es cuadrático en el peor caso, por eso es mejor que Quick Sort que siempre es lineal.
    c. No tiene ninguna ventaja, Heap Sort siempre es peor
    d. Ninguna de las anteriores

22. ¿Cuál de las siguientes afirmaciones es cierta?:
    a. El algoritmo Quick Sort es cuadrático ($O(n^2)$)en la mayoría de los casos
    b. El algoritmo de inserción directa es lineal (O(n)) cuando el array ya está ordenado
    c. El algoritmo Heap Sort es cuadrático en el peor caso
    d. El algoritmo Quick Sort es lineal en la mayoría de los casos

23. Supongamos que se ejecutan las siguientes sentencias, donde se llama a diferentes métodos de ordenación. La llamada `llenarArrayAleatorio (lista)` llena el array `lista` con números aleatorios.

. . .

```
int [] lista = new int[size];
llenarArrayAleatorio (lista);
quickSort (lista);
insercion (lista);
burbuja (lista);
...
```

¿Cuál de las siguientes afirmaciones es cierta?:

a. burbuja tardaría menos que inserción
b. quickSort siempre es el método más rápido
c. inserción es el método más rápido porque ordena un array ya ordenado
d. Burbuja es el método más rápido porque ordena un array ordenado dos veces

24. Dado el siguiente array (L) de enteros: (3, 1, -5, 2, 0, 30, 1, 2) (índices de 0 a 7) y aplicando el método de ordenación QuickSort para ordenación ascendente y utilizando como pivote el elemento de la derecha (pivote = L[der]), ¿qué afirmación es cierta?

a. No se puede usar como pivote el elemento L[der]
b. Los primeros elementos a intercambiar son L[0]=3; L[7]=2
c. Los primeros elementos a intercambiar son L[1]=1; L[5]=30
d. Ninguna de las anteriores.

25. Dado el siguiente array (L) de enteros: (2, 5, -1, 10, 3, 0, 14, 5) (índices de 0 a 7) se aplica el método de Inserción Directa para ordenación ascendente. Una vez que se ha ejecutado la segunda iteración del bucle externo (i=3) el array queda de la siguiente forma:

a. 0, -1, 2, 5, 10, 3, 14, 5
b. 2, -1, 5, 3, 0, 5, 10, 14
c. -1, 0, 2, 5, 10, 3, 14, 5
d. -1, 2, 5, 10, 3, 0, 14, 5

26. En el siguiente método se implementa *con algunos errores* el algoritmo Quick-Sort para ordenación descendente:

```
1. public  void quickSort(double [] v, int izq, int der){
2.     int i,j;
3.     double pivote;
4.     int aux;
5.     pivote = v[(izq+der) / 2];
6.     i = izq; j = der;
7.     do{
8.         while (v [i] > pivote) i++;
9.         while (v [j] < pivote) j--;
10.            if (i<=j) {
11.                aux = v [i];
12.                v [i] = v [j];
13.                v [j] = aux;
14.                i++;
15.                j--;
16.            } // end del if
17.        } while (i<=j);
```

```
18.         if (izq < j) quickSort(v,izq,j); // llamadas recursivas
19.         if (i<der) quickSort(v,i,der);
20.     }
```

Indicar dichos errores (puede haber varias respuestas verdaderas):

a. Línea 2. Las variables i y j tienen que ser double

b. Línea 18. Debe ser: `if (izq <= j) quickSort(v,izq,j);`

c. Líneas 8 y 9: Deben ser

```
while (v [i] < pivote) i++;
while (v [j] > pivote) j--;
```

d. Línea 4: La variable aux tiene que ser double

27. Aplicamos Quick-Sort ascendente a la siguiente secuencia de enteros: (3, 1, 0, 2, 0, -2, -5, 0), tomando como pivote el valor (0). Cuando se cruzan los índices por primera vez, se obtienen las siguientes particiones:

a. (0,1,0) (2, 3, -5, -2, 0)

b. (0,-5, -2, 0) (2, 0, 1, 3)

c. (0, -5, -2, 0, 0) (1, 2, 3)

d. (-5, -2, 0) 0 (0, 1, 2, 3)

28. Disponemos de la siguiente secuencia de enteros: (2, 5, -1, 9, 10, 4, 8, 0). Si queremos ordenarla de menor a mayor con el método Heap Sort, inmediatamente después de la construcción inicial del montículo la secuencia sería:

a. -1, 0, 2, 4, 5, 8, 9, 10

b. -1, 0, 2, 5, 10, 4, 8, 9

c. 10, 9, 8, 5, 4, 2, 0, -1

d. 10, 9, 8, 2, 5, 4, -1, 0

Nota: debe utilizarse un max-heap

29. Disponemos de la siguiente secuencia de enteros: (2, 5, -1, 9, 10, 4, 8, 0). Si queremos ordenarla de mayor a menor con el método Heap Sort, inmediatamente después de la construcción inicial del montículo la secuencia sería:

a. -1, 0, 2, 5, 10, 4, 8, 9

b. -1, 0, 2, 4, 5, 8, 9, 10

c. 10, 9, 8, 5, 4, 2, 0, -1

d. 10, 9, 8, 2, 5, 4, -1, 0

Nota: debe utilizarse un min-heap

30. ¿Cuál es el principal objetivo del montículo o heap en el caso de ordenación ascendente?

a. Seleccionar el elemento menor de todo el array en la raíz del árbol, que corresponde a la posición 0.

b. Seleccionar el elemento mayor de todo el array en la raíz del árbol, que corresponde a la posición 0

c. Seleccionar el elemento menor de todo el array en la raíz del árbol, que corresponde a la última posición del array.

d. Seleccionar el elemento mayor de todo el array en la raíz del árbol, que corresponde a la última posición del array.

Práctica 3

1. En el paso 1, el método `toString`
    a. recibe como parámetro el propio objeto
    b. es `private`
    c. devuelve `String`
    d. no devuelve nada, imprime en pantalla un `String`
2. En el paso 2, si nos confundimos y asignamos a RECORDSIZE el valor de 29 en lugar de 30, ¿qué ocurriría con los dos primeros registros?
    a. no afectaría a la escritura, se haría correctamente
    b. todos los registros se escribirán al inicio del fichero por lo que se sobrescribirá la información una y otra vez
    c. el primer registro se escribiría al inicio del fichero y el segundo empezando en el byte 29, o sea, sobrescribiendo el último byte de los 8 que ocupa el saldo
    d. el primer registro se escribiría al inicio del fichero y el segundo empezando en el byte 28
3. En el paso 3, ¿la variable `ULTIMO` podría ser final?
    a. si, dado que es un atributo de clase y todas las instancias tienen el mismo valor
    b. no, porque su valor se va a modificar a medida que se abren cuentas
    c. no, porque es un atributo de instancia que tiene un valor diferente para cada `Cuenta`
    d. si, porque de esta forma se permite que su valor se modifique
4. ¿Por qué en el paso 4 se produce una excepción?
    a. porque se intenta leer la `Cuenta` que está en la posición 60 (59 cuentas antes) del fichero y no hay tantas cuentas
    b. porque se intenta leer la `Cuenta` que está en la posición 59 (58 cuentas antes de ella) y no hay tantas en el fichero
    c. porque se intenta leer en el byte 59 y no hay tantos en el fichero
    d. porque se intenta leer en el byte 60 y no hay tantos en el fichero
5. Nos piden crear una excepción y lo habitual es:
    a. heredar de `RuntimeException`, esto es, crearla como no comprobada
    b. heredar directamente de `Exception`, o sea, crearla como comprobada
    c. que sea comprobada, esto es, que herede de `RuntimeException`
    d. que sea no comprobada, esto es, que herede directamente de `Exception`
6. En el paso 6, el método `Cuenta.dameULTIMO()`:
    a. debe ser `static` ya que no se aplica a ninguna cuenta, sino que funcionará aunque no exista ninguna
    b. es `static` ya que cada `Cuenta` tiene un valor diferente
    c. es `static` para que sólo pueda ser usada desde métodos de la misma clase

d. ninguna de las anteriores

7. En el paso 8, el bucle debe ser:
   a. de `1` a `ULTIMO`
   b. desde `0` a `ULTIMO`
   c. de `0` a `ULTIMO-1`
   d. de `1` a `ULTIMO-1`

8. En el paso 12, al hacer la clase `Cuenta` abstracta
   a. hay que mover la implementación de todas los métodos a las tres subclases
   b. es necesario cambiar el método `leeCuenta` ya que ahora ya no es posible instanciar una `Cuenta` directamente, sino que deberá instanciarse una de las tres subclases
   c. es necesario hacer públicos o protected los atributos de `Cuenta` para implementar el método `toString` en las subclases (o disponer de los correspondientes getters)
   d. es necesario tener al menos un método abstracto

9. En el paso 12
   a. es necesario implementar un segundo constructor para cada subclase (cada subclase tendrá finalmente dos constructores igual que la clase `Cuenta`)
   b. se usa el constructor que ya hay implementado en cada subclase (cada subclase tendrá finalmente un único constructor)
   c. no es necesario implementar ningún constructor ya que se usa el de por defecto
   d. es necesario volver a definir los atributos en las subclases

10. En el paso 16
    a. es necesario hacer público o protected el atributo `saldo` de `Cuenta` para que pueda ser modificado en el método reintegro de la subclase `CuentaAhorro`
    b. se debe invocar desde el método `reintegro` de la clase `CuentaAhorro` al método `reintegro` de la clase `Cuenta`, mediante `super.reintegro` una vez comprobado que no se sobrepasa el 25%
    c. la nueva excepción debe ser no comprobada
    d. ninguna de las anteriores