Prueba 2: Opción D: Programación orientada a objetos

Situación

En una ciudad funciona una compañía de autobuses locales. La compañía de autobuses usa rutas fijas con paradas señalizadas y, a veces, con refugios en ellas, para que la gente se resguarde del clima. Los pasajeros pagan al conductor la tarifa especificada para el trayecto cuando se suben al autobús.

En esta empresa hay muchos objetos, entre ellos:

Objeto	Descripción
Autobús	Vehículo físico que transporta *pasajeros* en una *ruta* específica y cuenta con un *conductor*.
Pasajero	Persona que viaja en un *autobús*.
Ruta	Serie de carreteras y calles que recorre el *autobús* desde su punto de partida hasta su destino.
Parada de autobús	Ubicación señalizada en una *ruta* en la cual la gente espera el *autobús*. Puede tener una simple indicación o un refugio con asientos.
Conductor	Persona cualificada que conduce un *autobús* por una *ruta* determinada.

Estos dos objetos ya se han definido para la compañía de autobuses:

RutaAutobús
Integer: ruta String: inicio
setRuta(Integer: ruta) setInicio(String: inicio) Integer getRuta() String getInicio() String toString()

Autobús
Integer: id String: conductor RutaAutobús: ruta
setId(Integer: id) setConductor(String: conductor) setRutaAutobús(RutaAutobús: ruta) Integer getId() String getConductor() RutaAutobús getRutaAutobús() String toString()

El método `toString()` devuelve una implementación de tipo String de un objeto.

En el código se implementan de la forma siguiente:

```
public class RutaAutobús
{
  private int ruta;
  private String inicio;
  public RutaAutobús(int r, String s)
  {
    setRuta(r);
    setInicio(s);
  }
  public void setRuta(int r){ ruta = r; }
  public void setInicio(String s){ inicio = s; }
```

```
  public int getRuta(){ return ruta; }
  public String getInicio(){ return inicio; }
  public String toString()
  {
    return "Ruta: " + ruta + " inicio: " + inicio;
  }
}



public class Autobús
{
  private int id;
  private String conductor;
  private RutaAutobús ruta;
  public Autobús(int i, String d, RutaAutobús r)
  {
    setId(i);
    setConductor(d);
    setRuta(r);
  }
  public void setId(int i){ id = i; }
  public void setConductor(String d){ conductor = d; }
  public void setRuta(RutaAutobús r){ ruta = r; }
  public int getId(){ return id; }
  public String getConductor(){ return conductor; }
  public RutaAutobús getRuta(){ return ruta; }
  public String toString()
  {
    return "Id autobús:" + id + " - " + conductor + ": " + ruta.toString();
  }
}
```

Temas troncales del NM y el NS

D1 (a) Explique el término *variable de parámetro*, usando un ejemplo del código. *[2 puntos]*

Una variable de parámetro es la que se pasa a un método, por ejemplo *setConductor(**String: Conductor**)* del objeto Autobús, siendo String: Conductor la variable.

(b) Describa **un** campo adicional que se podría incluir en la clase/objeto `RutaAutobús`. Incluya tipos de datos y ejemplos de datos. *[2 puntos]*

Se podría añadir un campo adicional 'fin' (destino). Podría ser una cadena de texto (p. ej. "Chelsea") o un número entero, siendo el número la ubicación correspondiente. Sería más fácil alterar el sistema si se usara un sistema de enteros, ya que muchas rutas podrían terminar en la misma ubicación (y así solo habría que corregir una entrada).

(c) Identifique la salida generada por el siguiente fragmento de código. *[2 puntos]*

```
Autobús autobús = new Autobús(1001, "N Prakesh",
new RutaAutobús(431, "Klang"));

System.out.println(bus.toString());
```

El fragmento de código produciría la salida siguiente:

```
ID Autobús: 1001 - N Prakesh: Ruta: 431 inicio:
Klang
```

Considere el siguiente fragmento de código.

```
private static final int MAX_BUSES = 12;

private Autobús[] autobuses = new Autobús[MAX_BUSES];

autobuses[0] = new Autobús(1001,
                    "N Prakesh",
                    new RutaAutobús(431, "Klang"));
autobuses[1] = new Autobús(1010,
                    "J Carey",
                    new RutaAutobús(342, "Tanglin"));
```

```
autobuses[2] = new Autobús(1014,
                           "H Lee",
                           new RutaAutobús(411, "Queenstown"));
autobuses[3] = new Autobús(1015,
                           "K Peters",
                           new RutaAutobús(319, "Jamaica
                           Street"));

mostrarConductoresAutobús(autobuses, 1010);
```

(d) Construya el método `mostrarConductoresAutobús(Autobús [] b, int n)` que muestre una lista con los conductores de todos los autobuses que tengan un número de ruta menor o igual que el valor del parámetro (`n`). *[6 puntos]*

```
public String mostrarConductoresAutobús(Autobús[]
b, int n) {
   for(int x = 0; x < MAX_BUSES; x ++) {
      if(b[x].getRuta().getRuta() < n) {
         System.out.println(b[x].getConductor() +
"\n"
      }
   }
}
```

La compañía desea realizar un seguimiento más detallado de sus conductores, incluyendo el nombre, el apellido y el número de empleado (éste es un número de cuatro dígitos).

(e) Construya un diagrama adecuado para el objeto `Conductor`. *[3 puntos]*

Conductor
String: nombre String: apellido Integer: num_empleado
setNombre(String n) setApellido(String a) setNumEmpleado(Integer e) String getNombre() String getApellido() Integer getNumEmpleado()

D2 En relación con el ejemplo del autobús:

(a) Esboce cómo se usa la encapsulación. *[2 puntos]*

La encapsulación se produce cuando un objeto usa métodos para restringir el acceso desde el exterior de su clase y para evitar que métodos externos modifiquen accidentalmente datos de la clase. Como ejemplo, el objeto Autobús contiene las variables privadas int id, String conductor y RutaAutobús, a las que no pueden acceder directamente otras clases/métodos.

(b) Esboce una desventaja de usar Diseño Orientado a Objetos. *[2 puntos]*

Construir un programa orientado a objetos es difícil y para muchas tareas menores es potencialmente innecesario.

(c) Explique cómo se podría beneficiar un equipo de programadores de un Diseño Orientado a Objetos. *[4 puntos]*

Un diseño orientado a objetos permite que un equipo divida un trabajo en varias tareas más pequeñas. Por ejemplo, cada miembro del equipo podría trabajar en un objeto individual (p. ej. uno podría trabajar en el objeto Autobús, mientras que el otro trabajaría en RutaAutobús).

Siempre y cuando se conozcan los fundamentos básicos de cada objeto, es decir, sus clases (sus variables y qué devuelven), los otros miembros del equipo pueden continuar con su trabajo.

Recuerde que una *parada* es uno de los muchos sitios **indicados** en una *ruta* en los que los *autobuses* paran para que suban o bajen *pasajeros*. Puede tener o no un **refugio** para que los *pasajeros* se resguarden del tiempo. La **distancia** en kilómetros desde la salida de la *ruta* es una información importante para la planificación.

(d) Diseñe el objeto ParadaAutobús usando un diagrama de Objeto simple. *[3 puntos]*

ParadaAutobús
String nombre Boolean refugio Double distancia
setNombre(String n) setRefugio(Boolean m) setDistancia(Double distancia) String getNombre() Boolean getRefugio() Double getDistancia()

(e) Sugiera cómo almacenar la información de una instancia de ParadaAutobús perteneciente a una RutaAutobús, poniendo ejemplos de datos y de fragmentos de código que muestren cómo se podría implementar. *[4 puntos]*

La clase RutaAutobús podría almacenarse como [??] una matriz de paradas de autobús, p. ej.:

```
ParadaAutobús[] matriz_paradas = new ParadaAutobús[x];
matriz_paradas[0] = new ParadaAutobús("Nombre", false, 3.50);
```

Las variables parámetro de ejemplo son las siguientes:
"Nombre" representa el nombre de la parada del autobús
false indica el valor booleano de la existencia o no de un refugio
3.50 es una cifra de tipo double que indica la distancia desde la salida

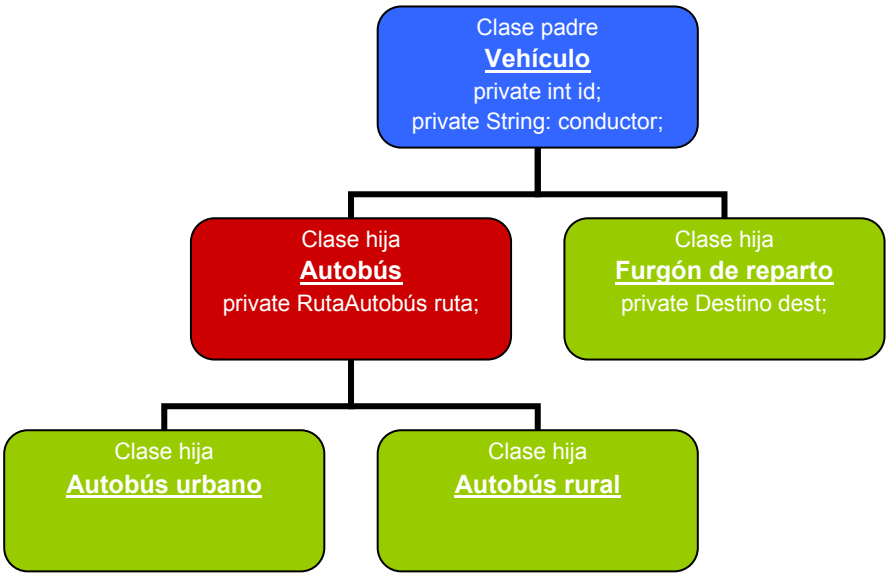
En primer lugar se crea la matriz ParadaAutobús, de tamaño x. Cada elemento se introduce posteriormente en la matriz (el elemento clave 0 se muestra anteriormente).

D3 La compañía va creciendo y ofrece más rutas de distintos tipos y decide usar tres tipos diferentes de vehículos:

- Un autobús que funciona en rutas con mucho tráfico (el Autobús Urbano) con un solo conductor.
- Un autobús más pequeño que opera en rutas más largas, que lleva una persona adicional para cobrar los billetes.
- Un furgón que se puede usar para realizar repartos de mercancías pesadas en lugar de pasajeros, el Furgón de Reparto: lleva un copiloto y un ayudante.

Estos vehículos tienen características comunes, como el *conductor*, y otros elementos que los distinguen. Por ejemplo, los autobuses urbanos y rurales operarán en una *ruta* fija mientras que el Furgón de Reparto lleva mercancías a *destinos* especificados (como fábricas u otros negocios).

(a) Construya diagramas para mostrar cómo se podría volver a diseñar la clase `Autobús` para implementar la herencia. *[8 puntos]*



(b) Explique la ventaja de usar la herencia en esta situación. *[4 puntos]*

Debe evitarse la repetición: Estructurar el diseño de esta forma permitiría que cada vehículo heredase las mismas cualidades fundamentales (como ID y conductor) de la clase padre a la vez que proporciona propiedades individuales a través de clases hijas separadas. Los autobuses rurales y urbanos podrían compartir propiedades adicionales y, tal vez, de esta forma se podría crear una clase 'Autobús' adicional que almacenara dichas propiedades. No tener código duplicado es especialmente útil cuando se encuentran errores en un programa.

Se requiere un método de las subclases que devuelva el número de empleados por vehículo.

(c) Esboce cómo se podría aplicar el polimorfismo en este diseño. *[3 puntos]*

El polimorfismo es una técnica que permite usar el mismo nombre para métodos que funcionan de forma ligeramente diferente. Cada clase debe contener un método contarEmpleados. La clase de vehículos puede contener un método contarEmpleados que devuelva uno. La clase de autobuses rurales puede reemplazar a este método y devolver dos.

Ampliación del NS

D4 La compañía de autobuses decide realizar una simulación en una ruta concreta para ver qué ocurre cuando varios autobuses inician la ruta a horas distintas, con un intervalo preestablecido. Se usará una cola para almacenar las instancias de `Autobús`.

(a) Identifique **tres** características de una cola que la hacen adecuada para este propósito. *[3 puntos]*

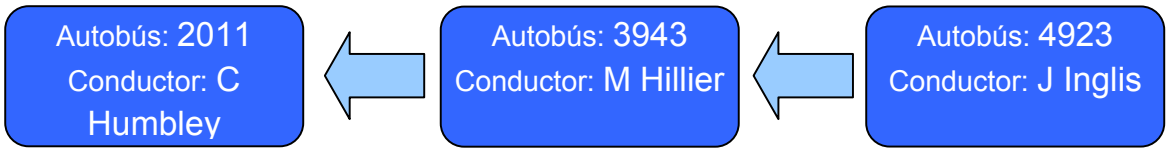
Una cola utiliza una estructura Primero en entrar, primero en salir (FIFO) que es adecuada para la tarea.

(b) Construya un diagrama de cómo quedaría la cola después de ejecutarse el siguiente código. *[3 puntos]*

```
public class SimAutobús
{
  private LinkedList<Autobús> colaAutobús;

  public static void main(String[] args){ new SimAutobús(); }
  public SimAutobús()
  {
    // Crea una nueva LinkedList para las instancias de Autobús
    colaAutobús  = new LinkedList<Autobús>();
    BusRoute ruta = new BusRoute (903, "Nerang Creek Road");
    Autobús autobús1 = new Bus(2011, "C Humbley", ruta);
    Autobús autobús2 = new Bus (3943, "M Hillier", ruta);
    Autobús autobús3 = new Bus (4923, "J Inglis", ruta);
    colaAutobús.addFirst(autobús1);
    colaAutobús.addFirst(autobús2);
    colaAutobús.addFirst(autobús3);

  }
}
```



Recuerde que el método de la clase `remove(int index)` de la clase `LinkedList` elimina el elemento de la posición especificada de la lista y que el método `size()` devuelve el número de elementos de una lista.

(c) Construya un método `eliminarAutobús(int posición)` que extraiga el autobús situado en la posición indicada de la cola y devuelva `true` si la acción finaliza correctamente o `false` si falla. *[4 puntos]*

```
private boolean eliminarAutobús(int posición) {
   if(posición < colaAutobús.size() && posición > -1) {
      colaAutobús.remove(posición);
      return true;
   }
   else { return false; }
}
```

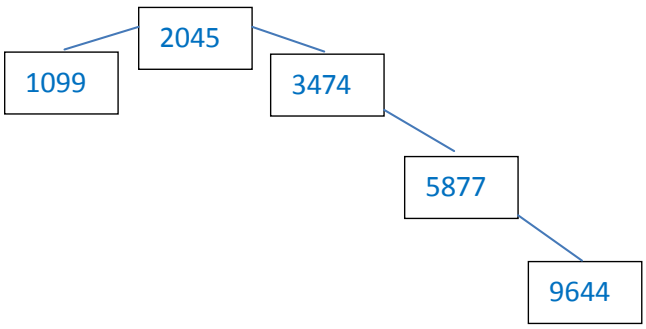
Una empresa de gran tamaño podría tener cientos de autobuses en ruta. Todos tienen un identificador que se almacena en la instancia de `Autobús`.

(d) Explique cómo se podría usar un árbol binario para almacenar esos identificadores de modo que se pueden localizar rápidamente (si existen) mediante una búsqueda. *[3 puntos]*

Un árbol binario es un método usado para reducir el tiempo necesario para buscar un valor concreto en una lista. El árbol ordena valores concretos a la izquierda y a la derecha dado un elemento inicial (p. ej. los valores de la izquierda son menores, los valores de la derecha son mayores). Esto hace que la búsqueda sea más eficiente, especialmente cuando existe una gran cantidad de datos.

El árbol contiene los identificadores 2045, 3474, 5877, 1099 y 9644.

(e) Dibuje un diagrama de un árbol binario ordenado que contenga esos valores, asumiendo que se insertan en el orden indicado. *[5 puntos]*



En un árbol binario, un nodo se puede insertar de forma iterativa o recursiva.

(f) Identifique **dos** desventajas del algoritmo recursivo. *[2 puntos]*

- Un algoritmo recursivo es más difícil de escribir
- Requiere más espacio de pila y puede generar un error de desbordamiento