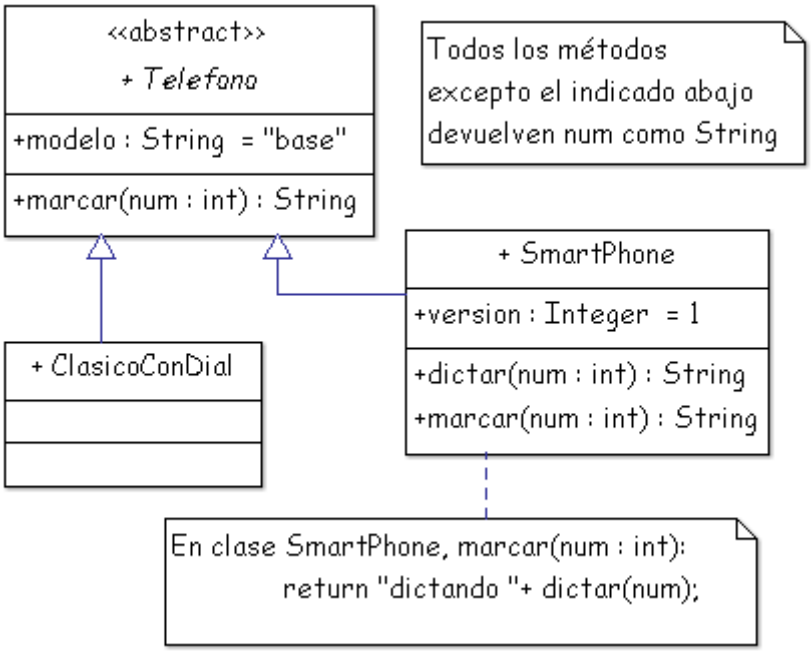
Prueba 1 de Evaluación Continua
Análisis y Diseño de Software (2018/2019)

Entrega cada ejercicio en hojas separadas

Ejercicio 1: (*3 puntos*)

El código Java dado abajo se basa en una implementación correcta del diagrama de clases de la figura, pero es posible que dicho código contenga errores en la utilización de esas clases.

Se pide:
Para cada línea numerada con comentarios en el código, indicar y justificar **si tiene algún error** distinguiendo si trata de errores de compilación o de ejecución; y **mostrar la salida por consola**, si la hay.




```
        ClasicoConDial ccd = new ClasicoConDial(); // No hay error
        SmartPhone smrtph = new SmartPhone();  // No hay error
/*  1 */ Telefono tccd = new ClasicoConDial();
/*  2 */ Telefono tsmrt = new SmartPhone();
/*  3 */ new SmartPhone().marcar(111);
/*  4 */ new Telefono().marcar(222);
/*  5 */ System.out.println("ccd version " + ccd.version );
/*  6 */ System.out.println("ccd modelo " + ccd.modelo );
/*  7 */ System.out.println("ccd modelo "+ ((Telefono) ccd).modelo );
/*  8 */ System.out.println("tsmrt version:" + tsmrt.version );
/*  9 */ System.out.println("tccd version:"+((SmartPhone)tccd).version );
/* 10 */ System.out.println( tsmrt.marcar(444) );
/* 11 */ System.out.println( ((SmartPhone) tsmrt).dictar(777) );
/* 12 */ System.out.println( tsmrt.dictar(555) );
/* 13 */ System.out.println( ((Telefono) smrtph).marcar(666) );
/* 14 */ System.out.println( ccd.marcar(888) );
/* 15 */ System.out.println( ((SmartPhone) ccd).marcar(999) );
```

Prueba 1 de Evaluación Continua
Análisis y Diseño de Software (2018/2019)

Solución Ejercicio 1:

La solución pedida es sólo la parte recuadrada. El resto es complemento para poder probar y ejecutarlo.

```
abstract class Telefono {
	public String modelo = "base";
	public String marcar(Integer num) { return num.toString(); }
}
class ClasicoConDial extends Telefono {
}
class SmartPhone  extends Telefono {
	public Integer version = 1;
	public String dictar(Integer num) { return num.toString(); }
	@Override public String marcar(Integer num) { return "dictando " + dictar(num); }
}

public class Ejercicio1 {
	public static void main(String[] args) {
	// indicar y justificar errores distinguiendo los de compilación y los de ejecución,
	// e indicar la salida que genera cada lína (en caso de que se imprima algo).
		ClasicoConDial ccd = new ClasicoConDial(); // No hay error
		SmartPhone smrtph = new SmartPhone();     // No hay error
	/*  1 */ Telefono tccd = new ClasicoConDial();
	/*  2 */ Telefono tsmrt = new SmartPhone();
	/*  3 */ new SmartPhone().marcar(111);
	/*  4 */ // new Telefono().marcar(222);
                                          // Error compilación: Telefono es abstract
	/*  5 */ // System.out.println("ccd version " + ccd.version );
                                         // Error compilación: ClasicoConDial no tiene version
	/*  6 */ System.out.println("ccd modelo " + ccd.modelo ); // imprime: ccd modelo base
	/*  7 */ System.out.println("ccd modelo "+ ((Telefono) ccd).modelo );
                                           // OK, casting OK, imprime ccd modelo base
	/*  8 */ // System.out.println("tsmrt version:" + tsmrt.version );
                                           // Error compilación: Telefono no tiene version
	/*  9 */ // System.out.println("tccd version:"+((SmartPhone)tccd).version );
                                                  // Error en ejecución: Casting inválido
	/* 10 */ System.out.println( tsmrt.marcar(444) );                // imprime dictando: 444
	/* 11 */ System.out.println( ((SmartPhone) tsmrt).dictar(777) );  // imprime 777
	/* 12 */ // System.out.println( tsmrt.dictar(555) );
                                                // Error compilación: Telefono no tiene dictar
	/* 13 */ System.out.println( ((Telefono) smrtph).marcar(666) );    // imprime dictando: 666
	/* 14 */ System.out.println( ccd.marcar(888) );                    // imprime: 888
	/* 15 */ // System.out.println( ((SmartPhone) ccd).marcar(999) );
                                                   // Error compilación: Casting inválido
	}
}
/* Salida obtenida:
ccd modelo: base
ccd modelo: base
dictando 444
777
dictando 666
888
*/
```

Prueba 1 de Evaluación Continua
Análisis y Diseño de Software (2018/2019)

Ejercicio 2: Diagrama de clases (*3.5 puntos*)

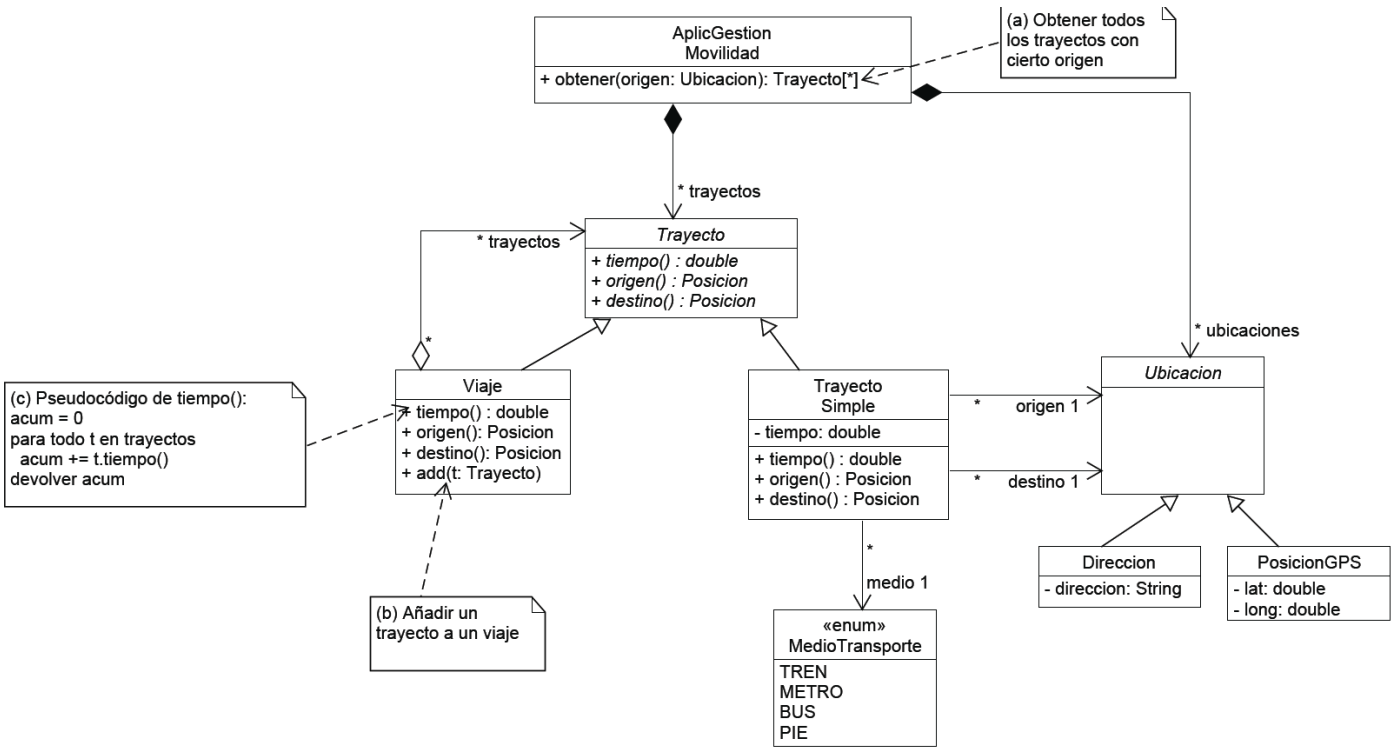
Nos piden realizar una aplicación para la gestión de la movilidad urbana en transporte público. El objetivo de la aplicación es que los usuarios puedan añadir y consultar el tiempo que se tarda en realizar trayectos en diversos medios de transporte.

Un trayecto simple tiene una duración en minutos, implica el uso de un medio de transporte (a pie, en tren, metro o bus), y tiene un origen y un destino. El origen y el destino pueden venir descritos bien por una dirección o por una posición GPS, descrita por una latitud y una longitud (que son números reales).

En la práctica, los viajes suelen implicar la realización de trayectos en distintos medios de transporte. De esta manera, un viaje contiene uno o más trayectos simples, u otros viajes previamente construidos. Tanto para los viajes como para los trayectos simples, es útil obtener su origen, destino y tiempo (donde el tiempo de un viaje es la suma del tiempo de los trayectos simples o viajes que contiene).

Se pide:

a. Realiza el diagrama de clases que describe la parte del diseño descrita arriba. (**2.5 puntos).**
   En este diagrama NO se incluirán constructores, getters ni setters, pero sí los métodos necesarios para cumplir los requisitos del segundo y tercer párrafos.
b. Incluye métodos en las clases diseñadas para: (**0.5 puntos).**
   b.1 Añadir viajes y trayectos simples a un viaje.
   b.2 Obtener todos los viajes y trayectos simples que tienen un cierto origen.
c. Describe en pseudocódigo el método para calcular el tiempo de un viaje (**0.5 puntos).**



Prueba 1 de Evaluación Continua
Análisis y Diseño de Software (2018/2019)

Ejercicio 3: Programación en Java (*3.5 puntos*)

Nos piden realizar una aplicación para la gestión de un almacén de artículos de ocasión. El almacén gestiona productos, que tienen un nombre, un precio en euros, y un descuento que puede ser bien un porcentaje configurable del precio total, o bien una cantidad fija de euros. Los descuentos tienen un texto que describe la promoción. Una vez creado un producto, no se puede modificar ni su precio ni su descuento.

Utilizando principios de orientación a objetos, codifica en Java las clases necesarias para que el siguiente programa dé la salida de más abajo.

(**Nota**: *por simplicidad,* ***puedes ignorar*** *el control de errores derivados de porcentajes de descuento negativos, mayores que el 100%, o bien descuentos mayores que el precio del producto*).

```
package productos;
public class Almacen {
  public static void main(String[] args) {
    // p1 es un Producto que vale 150.0€, y tiene 15.0% de descuento por promoción “sin IVA”
    Producto p1 = new Producto("Lámpara de pie", 150.0,
                               new DescuentoPorcentaje("Sin IVA", 15.0));
    // p2 es un Producto que vale 90.0€, y tiene 10.0€ de descuento por promoción “liquidación”
    Producto p2 = new Producto("Cubertería 50 piezas", 90.0,
                                new DescuentoFijo("Liquidación", 10.0));
    System.out.println("Productos en almacén:\n "+p1+"\n "+p2);
    System.out.println("Precio más alto: "+Producto.mayorPrecio());
  }
}
```

Salida Esperada:

```
Productos en almacén:
 Lámpara de pie precio: 127.5 con promoción: Sin IVA
 Cubertería 50 piezas precio: 80.0 con promoción: Liquidación
Precio más alto: 127.5
```

Prueba 1 de Evaluación Continua
Análisis y Diseño de Software (2018/2019)

Todas la clases están ubicadas en el paquete `productos`

```
public abstract class Descuento {
  protected double desc;
  protected String razonDesc;

  public Descuento(String r, double d) { this.razonDesc = r; this.desc = d; }
  public abstract double calculaPrecio(double d);
  public String toString() { return razonDesc; }
}

public class DescuentoFijo extends Descuento {
  public DescuentoFijo(String r, double d) { super(r, d); }
  @Override public double calculaPrecio(double d) { return d - this.desc; }
}

public class DescuentoPorcentaje extends Descuento {
  public DescuentoPorcentaje(String r, double d) { super(r, d); }
  @Override public double calculaPrecio(double d) { return d-d*this.desc/100.0; }
}

public class Producto {
  private String nombre;
  private double precio;
  private Descuento desc;
  private static double mayorPrecio = 0;

  public Producto(String n, double p, Descuento d) {
    this.nombre = n;
    this.precio = p;
    this.desc = d;
    mayorPrecio = d.calculaPrecio(this.precio) > mayorPrecio ?
                  d.calculaPrecio(this.precio) : mayorPrecio;
   }
   public static double mayorPrecio() { return mayorPrecio; }
   public String toString() {
       return this.nombre+" precio: "+desc.calculaPrecio(this.precio)+
              " con promoción: "+this.desc;
   }
}
```