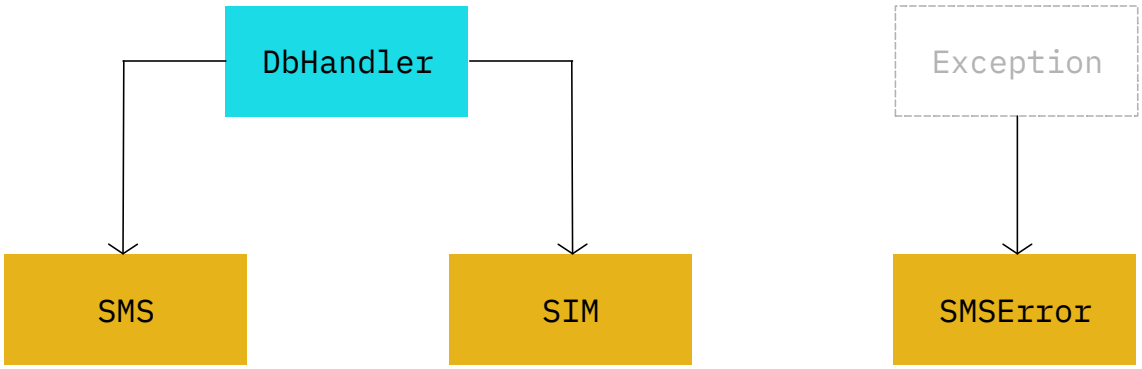
SMS

Este ejercicio gira en torno a la gestión de **mensajes SMS** desarrollando un programa en **Python** mediante el paradigma de **programación orientada a objetos** junto con **acceso a datos**.

La idea es simular el **envío** y **recepción** de *SMS* a traves de una pequeña base de datos.

1. Diagrama de clases

A continuación se presentan las clases que deben ser implementadas (junto con su **herencia**):



2. Base de datos

A continuación se especifican las tablas que componen la base de datos y su estructura:

Tabla `activity`

Representa la actividad de envío de SMS.

Columna	Tipo	Descripción
`id`	Entero (PK)	Clave primaria
`sender`	Texto	Teléfono del remitente
`recipient`	Texto	Teléfono del destinatario
`message`	Texto	Mensaje

Ejemplo de registro:
`|3|'634111908'|'654867721'|'Python 3.14 is awesome'|`

Tabla `access`

Representa las credenciales de acceso al teléfono.

Columna	Tipo	Descripción
`phone_number`	Texto (PK)	Número de teléfono
`pin`	Texto	PIN de acceso
`puk`	Texto	PUK de desbloqueo

Ejemplo de registro:
`|'634111908'|'3241'|'9058'|`

3. Clase `DbHandler`

Representa una entidad abstracta que contiene funcionalidades para manejo de bases de datos y de la que heredan otras clases.

Implementa, *al menos*, los siguientes **métodos**:

```
def __init__(self, db_path: str = DB_PATH):
```

- Constructor de la clase.
- Crea el atributo `con` ⇒ conexión a la base de datos (especificando factoría `Row`).
- Crea el atributo `cur` ⇒ cursor a la base de datos.
- En este punto **no hay que almacenar nada en la base de datos**.

```
def create_db(self) -> None:
```

- Crea la base de datos y sus correspondientes tablas.
- **No cierres** la conexión a la base de datos.

4. Clase `SMS`

Representa un SMS.

Implementa, *al menos*, los siguientes **métodos**:

```
def __init__(self, sender: str, recipient: str, message: str):
```

- Constructor de la clase.
- Esta clase hereda de `DbHandler`...
- Crea los atributos homónimos a los parámetros.
- En este punto **no hay que almacenar nada en la base de datos**.

`def send(self) -> None:`

- Simula el envío del SMS `self` mediante una operación en la base de datos.
- Inserta todos los campos del SMS en la tabla `activity`.

`def __str__(self):`

- Representa un objeto de tipo `SMS` de la siguiente forma:

```
From: <remitente>
To: <remitente>
---
<mensaje>
```

5. Clase `SIM`

Representa una tarjeta SIM.

Implementa, *al menos*, los siguientes **métodos**:

`def __init__(self, phone_number: str):`

- Constructor de la clase.
- Esta clase hereda de `DbHandler`...
- Crea el atributo `phone_number` desde el argumento.
- Crea el atritubo `unlocked` (`bool`) que indique se la tarjeta se ha desbloqueado o no.
- En este punto **no hay que almacenar nada en la base de datos**.

`def unlock(self, pin:str, *, puk: str = ''): -> None`

- Intenta desbloquear la tarjeta SIM comprobando PIN y PUK que se pasan (mediante una consulta a la base de datos).
- Hay que **actualizar** el atributo `unlocked` de la siguiente manera:
  - Si el PIN aportado es correcto se desbloqueará la SIM.
  - Si el PIN aportado no es correcto, se procederá a comprobar el PUK. En caso de que el PUK sea correcto se desbloqueará la SIM.
  - En caso de que el teléfono no exista, no se podrá desbloquear la SIM de ninguna manera.


```
@staticmethod
def unlock_required(method):
```

- Decorador para comprobar si la tarjeta SIM está desbloqueada.
- En caso de no estarlo, habrá que lanzar una excepción de tipo `SMSError` con el mensaje SMS is locked
- Recuerda que la excepción recibe en su constructor tanto el mensaje de error como el objeto actual de tipo `SIM`.

```
@unlock_required
def send_sms(self, *, recipient: str, message: str) -> None:
```

- Realiza (simula) en envío de un SMS.
- Habrá que construir un objeto `SMS` y enviar el mensaje mediante los argumentos recibidos.
- Si `recipient` no tiene un formato válido de *teléfono* habrá que lanzar una excepción de tipo `SMSError` con el mensaje Recipient has invalid phone format
- Un número de teléfono válido es aquel que empieza por 6 o 7 y contiene un total de 9 dígitos. Cuidado porque también puede aparecer un prefijo con 2 dígitos:
  - ✓ +34 678543559
  - ✓ 678543559
  - ✓ +51 732869123
  - ✓ 732869123
- Recuerda que la excepción recibe en su constructor tanto el mensaje de error como el objeto actual de tipo `SIM`.

```
@unlock_required
def get_sms(self, sent: bool = True):
```

- **Función generadora** que devuelve objetos de tipo `SMS`.
- El comportamiento es el siguiente:
  - Si `sent` es verdadero, se devuelven los SMS enviados por el número de teléfono actualmente desbloqueado.
  - Si `sent` es falso, se devuelven los SMS recibidos por el número de teléfono actualmente desbloqueado.

6. Clase `SMSError`

Representa un error en la gestión de SMS.

Implementa, *al menos*, los siguientes **métodos**:


```
def __init__(self, message: str, db_handler: DbHandler):
```

- Constructor de la clase.
- El argumento `message` es el mensaje que debe procesar la excepción.
- Esta clase hereda de `Exception`...
- Cierra la conexión a la base de datos.

