```python
import  scipy.constants as spc
import  numpy as np
from    matplotlib import  pyplot as plt
from    typing import Dict, Tuple, Optional

class Task1:

    __slots__ = ["E_0", "E_F", "effMass", "channelWidth"]

    def __init__(self,
                 relativeFermiEnergy: float,
                 effectiveMass: float,
                 channelWidth: float,
                 groundState: float = 0
                 ) -> None:
        """Sets up the system from task 1.

        Args:
        relativeFermiEnergy (float): How many electron volts the fermi
            energy is above the ground state in the z-direction.
        effectiveMass (float): Effective mass of an electron in the system as a percentage of
            the electron mass of a free electron.
        channelWidth (float): The width of the channel in the y-direction.
        groundState (float, optional): The ground state energy in eV in the z-direction.
Defaults to 0
            for reference.
        """

        # Ground state energy in z-direction in electron volts.
        # Defaults to 0 for reference.
        self.E_0 = groundState

        # Fermi energy is 90 meV above ground
        # state energy in z-direction.
        self.E_F = self.E_0 + relativeFermiEnergy

        # Effective mass of an electron in the system in kg
        self.effMass = effectiveMass * spc.electron_mass

        # Width of the channel in y-direction in meters
        self.channelWidth = channelWidth

    def E_x(self, k_x: np.array) -> np.array:
        """
        Energy eigenvalues for psi(x) as a function of k_x.
        """
        return (
                ( (spc.hbar**2 / spc.electron_volt) * k_x**2 )
                /
                ( 2 * self.effMass )
               )

    def E_y(self, n: int) -> float:
        """
        Energy eigenvalues of psi(y) as a function of
        the quantum number n.
        """
        return  (
                 ( np.pi**2 * (spc.hbar**2 / spc.electron_volt) * n**2 )
                 /
                 ( 2 * self.effMass * self.channelWidth**2 )
                )

    def E_n(self, k_k: np.array, n_min: int, n_max: int) -> Dict:
        """
        Returns a dictionary with the energy bands of the system.
        """
```

```
    bands = {}

    for n in range(n_min, n_max + 1):
        bands[f"n = {n}"] = ( self.E_0 + self.E_x(k_k) + self.E_y(n) )

    return bands

def plotEnergyBands(self, band_min: int, band_max: int) -> None:
    """Plots the energy bands of the system.

    Args:
        band_min (int): The lowest band to be included.
        band_max (int): The highest band to be included.
    """

    k_x: np.ndarray = np.linspace(-1e9, 1e9, 1000)
    bands: dict = self.E_n(k_x, band_min, band_max)

    closestIndex = min(range( len(bands["n = 1"]) ), key=lambda i: abs(bands["n = 1"][i]
        - (self.E_F + 30e-3) ))
    x_max = k_x[closestIndex]

    self._plot(k_x, bands, "Energy bands", "$k_x$ [$m^{-1}$]", "Energy [eV]")

    plt.axhline(self.E_F, linestyle="dashed", color="red", label="$E_F$")
    #plt.axhline(self.E_0, linestyle="dashdot", color="green", label="$E_F$")
    plt.ylim(top= self.E_F + 30e-3, bottom = 0 )
    plt.xlim(left = -x_max, right=x_max)
    plt.legend()

    plt.show()

def _plot(self,
          xVal: np.array,
          yVals: dict,
          title: str,
          xLabel: str,
          yLabel: str,
          figsize: Tuple[int, int] = (10,6),
          savePath: Optional[str]  = None
          ) -> None:

    font = {'family': 'serif', 'color': 'darkred',
            'weight': 'normal', 'size': 16,}

    # Create the plot
    plt.figure(figsize=figsize)
    plt.title(title, fontsize=35, fontdict=font, y=1.05)
    plt.xlabel(xLabel, fontsize=30, fontdict=font)
    plt.ylabel(yLabel, fontsize=30, fontdict=font)

    for key in yVals:
        plt.plot(xVal, yVals[key], label=key)

    plt.xticks(fontsize=20)
    plt.yticks(fontsize=20)
    plt.grid(True, which='both', linestyle='--', linewidth=0.5)
    plt.legend(fontsize=10)

    if savePath is not None:
        plt.savefig(savePath)

def bandsBelowFermiEnergy(self) -> int:
    """
    Returns an integer representing the nr. of bands
    below the fermi energy in the system.
```

```
        """

        n = 1
        energy = 0
        while energy < self.E_F:

            bandMinimum = self.E_y(n)
            if bandMinimum <= self.E_F:
                n += 1
            else:
                return n -1

    def getCurrent(self, V_sd: float) -> float:
        """
        Calculates the current through the channel based on the applied
        voltage between the source- and drain terminals.
        """
        current = (
            self.bandsBelowFermiEnergy() *
            ( (2 * spc.elementary_charge**2) / spc.h ) *
            V_sd
        )

        return current


if __name__ == "__main__":

    system = Task1(
        relativeFermiEnergy = 90e-3,
        effectiveMass = 0.097,
        channelWidth = 120e-9,
        groundState = 0
    )

    nrBands = system.bandsBelowFermiEnergy()
    print(f"Bands below the fermi energy: {nrBands}")
    print(f"Energy of band nr. {nrBands}: {system.E_y(nrBands)} eV")
    print(f"Current as a result of V_sd = 30 uV: {system.getCurrent(30e-6)} A")
    system.plotEnergyBands(band_min = 1, band_max = 19)
```