Here's the Python class for Principal Component Analysis:

Python

```
import numpy as np

class PrincipalComponentAnalysis:
    """
    Implements Principal Component Analysis (PCA) algorithm for dimension

    Args:
        n_components: (int, optional) Number of principal components to k
        decomposition_method: (str, optional) Decomposition method to use
    """

    def __init__(self, n_components=None, decomposition_method='eigen'):
        self.n_components = n_components
        self.decomposition_method = decomposition_method

    def fit(self, X: np.ndarray) -> None:
        """
        Fits the PCA model to the given data.

        Args:
            X: (np.ndarray) Data to be transformed.
        """
        # Check if data is a numpy array
        if not isinstance(X, np.ndarray):
            raise TypeError('Data must be a numpy.ndarray')

        # Check dimensions
        if X.ndim != 2:
            raise ValueError('Data must be a 2D matrix')

        # Check if n_components is an integer and greater than 0
        if self.n_components is not None and not isinstance(self.n_compon
            raise ValueError('n_components must be an integer greater tha

        # Check if decomposition_method is valid
        if self.decomposition_method not in ['eigen', 'svd']:
            raise ValueError('Invalid decomposition_method: must be eithe

        # Calculate mean and covariance matrix
        self.mean = np.mean(X, axis=0)
        self.covariance_matrix = np.cov(X.T)

        # Perform dimensionality reduction using specified method
        if self.decomposition_method == 'eigen':
            self.eigvals, self.eigvecs = np.linalg.eig(self.covariance_ma
        elif self.decomposition_method == 'svd':
            U, S, Vh = np.linalg.svd(self.covariance_matrix)
            self.eigvals = S**2
            self.eigvecs = Vh.T
```

```python
        # Sort eigenvectors by descending eigenvalues
        indices = np.argsort(self.eigvals)[::-1]
        self.eigvals = self.eigvals[indices]
        self.eigvecs = self.eigvecs[indices]

        # Truncate eigenvectors to desired number of components
        if self.n_components is not None:
            self.eigvals = self.eigvals[:self.n_components]
            self.eigvecs = self.eigvecs[:, :self.n_components]

    def transform(self, X: np.ndarray) -> np.ndarray:
        """
        Transforms the given data using the fitted PCA model.

        Args:
            X: (np.ndarray) Data to be transformed.

        Returns:
            np.ndarray: Transformed data in the lower-dimensional space.
        """

        # Check if data is a numpy array
        if not isinstance(X, np.ndarray):
            raise TypeError('Data must be a numpy.ndarray')

        # Check dimensions
        if X.ndim != 2:
            raise ValueError('Data must be a 2D matrix')

        # Standardize data
        Z = X - self.mean

        # Project data onto principal components
        transformed_data = Z @ self.eigvecs

        return transformed_data

    def fit_transform(self, X: np.ndarray) -> np.ndarray:
        """
        Fits the PCA model to the given data and transforms it simultaneo

        Args:
            X: (np.ndarray) Data to be fit and transformed.

        Returns:
            np.ndarray: Transformed data in the lower-dimensional space.
        """

        # Fit the model
        self.fit(X)

        # Transform the data
        return self.transform(X)
```