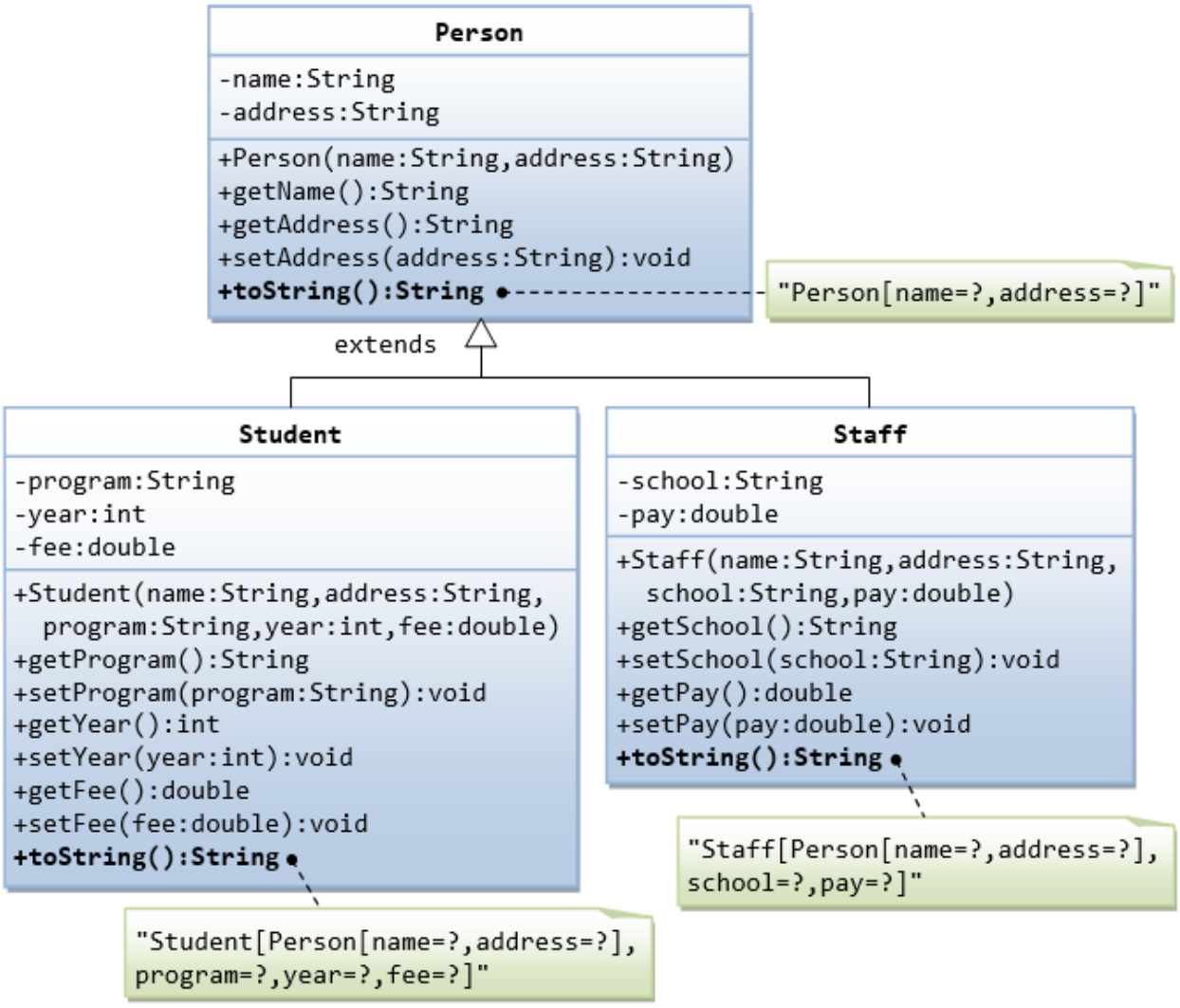
# PROGRAMMING 2

## Tutorial 4 - Inheritance

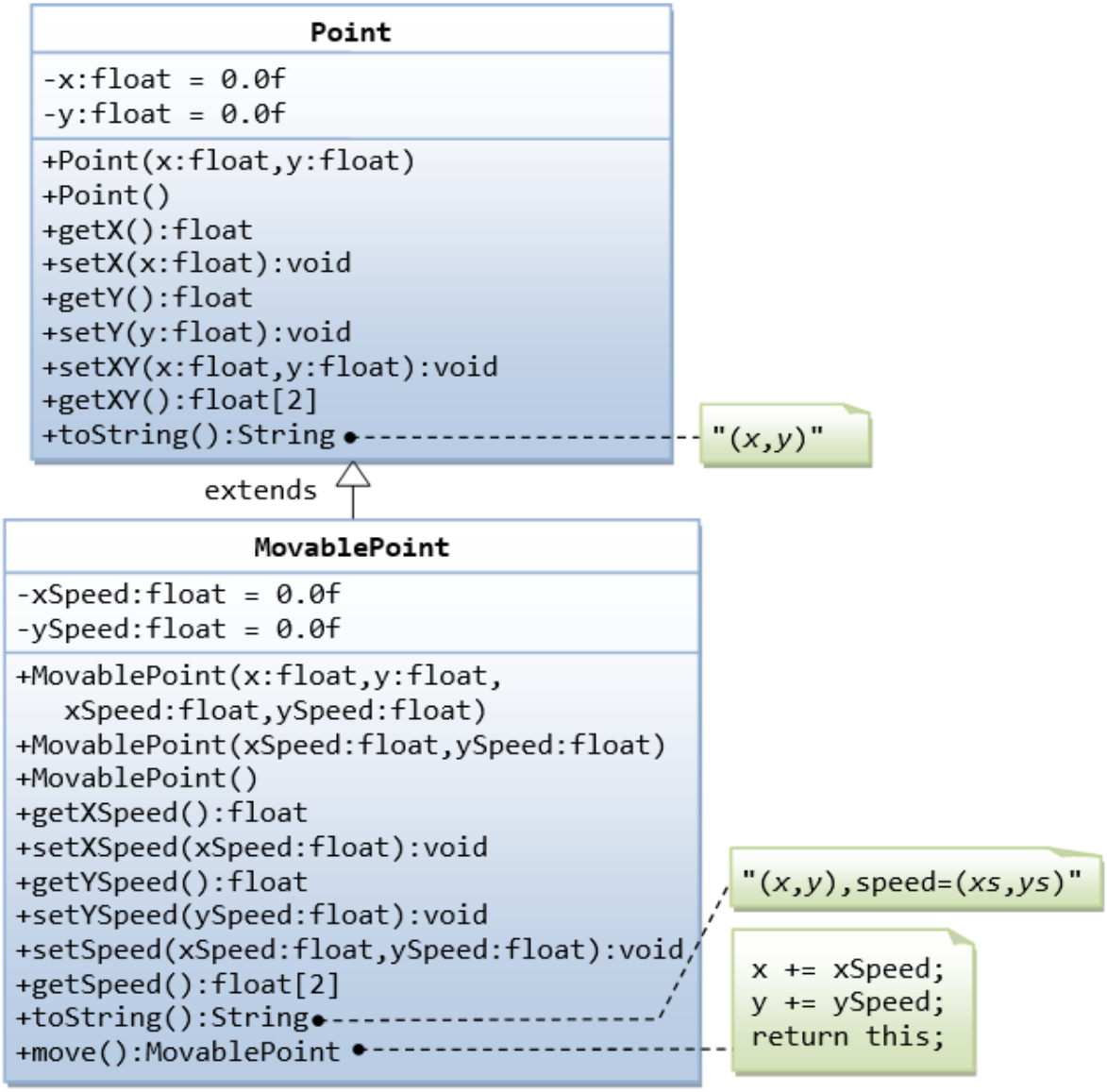
### Activity 1: Person

Implement the following classes defined as shown in the diagram. Note that you should mark all overridden methods with the annotation `@Override`.



## Activity 2: Point

Implement the following classes defined as shown in the diagram. Note that you should mark all overridden methods with the annotation `@Override`.




**Notes:**

- Floating-point literals are written with a decimal point. By default, a floating-point literal is treated as a **double** type value. For example, **5.0** is considered a **double** value, not a **float** value. You can make a number a **float** by appending the letter **f** or **F**, and you can make a number a **double** by appending the letter **d** or **D**. For example, you can use **100.2f** or **100.2F** for a **float** number, and **100.2d** or **100.2D** for a **double** number.
- A private data field cannot be accessed by an object from outside the class that defines the private field. However, a client often needs to retrieve and

modify a data field. To make a private data field accessible, provide a *getter* method to return its value. To enable a private data field to be updated, provide a *setter* method to set a new value. The getter method is also referred to as an *accessor* and a setter to a *mutator*.

- A getter method has the following signature:

  ```
  public returnType getPropertyName()
  ```

- If the **returnType** is **boolean**, the getter method should be defined as follows by convention:

  ```
  public boolean isPropertyName()
  ```

- A setter method has the following signature:

  ```
  public void setPropertyName(dataType propertyValue)
  ```

## Activity 3: Shape

In this exercise, you will be creating a Java program that models shapes using object-oriented programming principles. The program will include a superclass called `Shape`, which represents generic shapes, and two subclasses: `Circle` and `Rectangle`. Additionally, you will create a subclass of `Rectangle` called `Square`. The properties and methods defined as shown in the following diagram. Note that you should mark all overridden methods (denoted as **bold**) with the annotation `@Override`.

## Activity 4: SortingAlgorithm.java

In this exercise, your mission is to create a hierarchy of sorting integer arrays algorithms. You will start with a superclass called `IntSort`, which will serve as the base class for implementing various sorting algorithms. The class should have the following properties and methods:

- **Properties:**
  - `name`: a string representing the name of the sorting algorithm.
  - `array`: an integer array to be sorted.
  - `sortedArray`: the sorted array.
- **Methods:**
  - `sort()`: A method to sort an integer array in ascending order utilizing a specific sorting algorithm. The method should not return any value but store the sorted array in the `sortedArray` attribute of the class.
  - `isSorted()`: a method to determine if the array is sorted in ascending order. It should return `true` if the array is sorted, otherwise `false`.
  - `toString()`: a method to return a string representation of the sorted array.

Necessary methods such as getters, setters should be implemented.

Other than the super-class with a default sorting algorithm of your choice, you are required to implement at least two sub-classes of `IntSort` that implement two other sorting algorithms. Examples of sorting algorithms you can implement include **Selection Sort**, **Insertion Sort**, **Bubble sort**, **Quick sort**, **Merge sort**, etc. Examples of sub-class names: `IntSelectionSort`, `IntQuickSort`…

## Activity 5: Cryptography (optional)

In this exercise, you will implement a simple encryption. You will start with a superclass called `Crypto`, which will serve as the base class for implementing various algorithms. The class should have (at least) the following properties and methods:

- **Properties:**
  - `plainText`: a string representing the plain text to be encrypted.
  - `cipherText`: a string representing the encrypted text.
- **Methods:**
  - `encrypt()`: a method that performs the encryption. By default, it uses the Caesar algorithm provided in the `CaesarCipher.java` file. The method should not return any value but store the encrypted text in the `cipherText` attribute of the class.
  - `decrypt()`: a method that performs the decryption. It should use the same cryptography algorthm with the `encrypt()` method. The method should not return any value but store the plain text in the `plainText` attribute of the class.

Necessary methods such as getters, setters should also be implemented.

You are provided with the `SubstitutionCipher.java` file for reference. You have to create a sub-class of the `Crypto` class called `SubCrypto` which implements the Substitution Cipher algorithm by overriding suitable methods of its super-class.

## Submission

Submit a zip file containing all Java programs to this tutorial's submission box in the course website on FIT LMS.