Person programming task

This is an English version of **Objektstrukturen - Person-oppgave**, and is concerned with a **Person** class and the consistency of the child-parent association.

An important part of the implementation of associations is to ensure consistency, i.e. that objects are correctly linked to each other in both directions. An example is associations between children and parents in a family tree. In this programming task, you will implement a **Person** class with logic for correctly adding (and removing) children, as illustrated below.

The **Person** class must support a **name** (a String) and **gender** (a character, ‘F’ eller ‘M’), both of which must be provided in the constructor and not be settable later, and **mother**, **father** and **children**, which are **other Person** objects. The **Person** class must have the following methods for reading the state:

- **getName()** - returns the name of this **Person** object
- **getGender()** - returns the character representing the gender, either 'F' or 'M'
- **getMother()** - returns the **Person** object representing the mother, or null
- **getFather()** - returns the **Person** object representing the father, or null
- **getChildCount()** - returns the number of children of this **Person** object
- **getChild(int n)** - returns child number n (another **Person** object), or raises an appropriate exception if n is too big (or small)

The **Person** class has two sets of mutation methods, related to the two **roles** of each end of the **children-mother/father** association.

From the **children** perspective we have the following two methods:

- **addChild(Person)** - links this **Person** object to a child (another **Person** object). If this **Person** object is a **woman** then she should also become the child’s **mother**, and alternatively, if this **Person** object is a man then he should become the child’s **father**.
- **removeChild(Person)** - removes a link from this **Person** object to a child (another **Person** object). If this **Person** object is the **mother** of the argument, then the **mother** link must be removed, and alternatively, if this **Person** object is the **father** of the argument, then the **father** link must be removed.

From the **mother/father** perspective we have the following two methods:

- **setMother(Person)** - sets the argument (a woman) as this **Person** object’s **mother**. This **Person** object should also become a **child** of the argument.
- **setFather(Person)** - sets the argument (a man) as this **Person** object’s **father**. This **Person** object should also become a **child** of the argument.

Note that both sets of methods, **addChild/removeChild** og **setMother/setFather**, must include logic to handle the link the opposite direction, so **addChild/removeChild** must call **setMother/setFather** and vice versa, or have corresponding effect. This can be a bit tricky, since you both must ensure consistency and avoid infinitely nested calls (until you get a StackOverflowException).

The figures below illustrate some of the cases that must be handled, and which are tested by our tests.
The task is split in two parts, the first handles the children- and mother/father roles in isolation without considering consistency, while the other must ensure consistency.

The Exercise panel

Use the Exercise panel, open the `Person.ex` file (tests > objectstructures > Person.ex) in the panel before you start programming.

Part 1

- Implement the `addChild` and `removeChild` methods so the `getChildCount` and `getChild` methods work as expected. These methods only handle the children role.
- Implement the `setMother` and `setFather` methods so the `getMother` and `getFather` methods work as expected. These methods only handle the mother/father roles.

Test the methods by creating `Person` object structures corresponding to your own family. You’ll have to use all the three methods `addChild`, `setMother` and `setFather`. Try with at least three generations.

Part 2

Extend the methods to handle consistency. Test by creating the same `Person` object structures as in part 1, this time by only using `addChild` and by only using `setMother` and `setFather`.

Test code for this task can be found here: objectstructures/PersonTest.java. The original jextest code can be found here: objectstructures/Person.jextest.