Project with JML

Part 1

In the file `Taxpayer.java` the Java class is used in an information system at the tax office to record information about individuals. The tax office requires that the following properties hold:

1. Persons are either male or female.
2. Persons can only marry people of the opposite sex.
    a. An obvious property, easy to overlook: if person x is married to person y, then person y should of course also be married and to person x.

Your assignment is to

1. add *invariants* to the code to express the properties above, in JML syntax, plus any other sensible properties you can think of;
2. use OpenJML to detect possible violations, reported as *warnings*
3. for the Java methods in the complains, fix them by
    a. correcting the code, or
    b. adding a (sensible) precondition for the method, using a JML *requires* clause, or
    c. adding an invariant for another property that can help in the verification.

Hints

- ✔ Only look at the **first** warning that the tool produces, and ignore the others, until you have managed to eliminate this first one.
- ✔ In the methods `marry` and `divorce` in the class `Taxpayer` the code is not correct. The tool should detect it and complain about these methods after adding your invariants; you need to add a few lines of code to fix these methods, and possibly a precondition by means of a requires clause.
- ✔ It is easiest to introduce one invariant at a time, then fix the errors detected, and only then move on to the next one.
- ✔ If the tool produces some warnings, i.e., if the tool thinks that some property is not satisfied by the code, this can have several causes:
    - o There is an error in the code, which results in a violation in the Java code. For example, an assignment to some field may be missing.
    - o There is an error in the JML specification. For example, maybe you have written "and" in a specification where you meant "or", maybe you have written age > 18 when you meant age >= 18 in some constraint
    - o There may be some properties missing but needed by the tool for the verification. Note that the tool does not know any of the things that may be obvious to you -- for example, that if some person x is married to y then y is also married to x – and such properties may be needed for verification.
    - o In general, a warning might be caused by the fact that some property is too difficult for the theorem-prover to verify: an automated theorem-prover can only ever prove

properties of a certain, limited complexity. However, for the exercise here you should not run into this problem.

To stop the tool from complaining, you can

- ✗ correct the Java program; for the exercise here this should only involve adding some simple assignments to the offending method;
- ✗ correct the JML specification;
- ✗ specify some additional properties, either as a JML invariant or as JML precondition aka requires clause.

In the end, the tool should run without any complaints on the annotated code, meaning that it has succeeded in verifying that the code meets the formal specification.

Part 2

The tax system uses an income tax allowance (deduction):

1. Every person has an income tax allowance on which no tax is paid. There is a default tax allowance of 5000 per individual and only income above this amount is taxed.
2. Married persons can pool their tax allowance, as long as the sum of their tax allowances remains the same. For example, the wife could have a tax allowance of 9000, and the husband a tax allowance of 1000.

Add invariants that express these constraints, and, if necessary, fix/improve the code to ensure that they are not violated.

Part 3

The new government introduces a measure that people aged 65 and over have a higher tax allowance, of 7000. The rules for pooling tax allowances remain the same.

Add invariants that express these constraints, and if necessary fix/improve the code to ensure that they are not violated.