2021-22 Project with JML

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

1. Persons are either male or female.
2. Persons can only marry people of the opposite sex.
3. 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 (or ESC/Java) to detect possible violations, reported as *warnings*
3. for the Java methods in the complains, fix them by (in order of preference)
   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.
- The code of the methods `marry` and `divorce` in the class `Person` is not correct. The tool should detect it and complain about these methods once you have added 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:
  - 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.
  - 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
  - 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.
  - 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 exercises here you should not run into this problem.

To stop the tool from complaining, you can

- correct the Java program; for this exercise this should only involve adding some simple assignments to the offending method
- correct the JML specifications

- 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.

JML syntax

Below the syntax for Java and JML logical connectives, in order of decreasing precedence:

*logical connective *	*Java/JML syntax*
not	!
equality	==
inequality	!=
conjunction (and)	&&
disjunction (or)	||
implication	==> , <==
(in)equivalence	, <=!=>

In addition to this, you can use standard Java syntax, so you can say things such as

```
 this.spouse != null ==>
   (this.spouse.spouse.isFemale ==> this.isFemale)
```

Of course, you can write spouse instead of this.spouse, leaving this implicit. Note that in Java, as in most programming languages, references such as the spouse field, can be null. Often fields should not be null under certain circumstances, which can be expressed by an invariant of the form

```
 //@ invariant  .... ==> spouse !=null;
```

Often arguments of a method should not be null, which can be expressed by a precondition of the form

```
 //@ requires new_spouse != null;
 void marry (Person new_spouse) {
```

Part 2

The welfare system has rules to modify allowances based on marriage and age:

1. Every person receives a default subsidy of 500 until age 65.
2. After the age of 65, the default subsidy increases to 600 if unmarried
3. Married persons receive each the default subsidy reduced by 30%

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

General hints

Some hints to keep you out of trouble with the tool:

* Do not use method invocations inside methods for this exercise.
* Do not use the Java shorthand x+=10 for x = x+10.
* Whenever possible, split invariants than contain conjunctions into several smaller invariants. This improves feedback of the tool. So instead of

```
//@ invariant A && B;
```

  write

```
//@ invariant A;
//@ invariant B;
```

  The same goes for preconditions (aka requires clauses).
* If the tool complains of invariant violations, it produces a lot of feedback that is hard to interpret. The crucial information to look at is
  - the line number saying **which** invariant is violated, and
  - the line number saying **where** this invariant is violated.

  NB beware that calling a method on some object may break the invariant of another object; for instance, in this exercise, calling a method on a Person may break the invariant of his or her spouse.