Es soll eine Klasse `Person` testgetrieben entwickelt werden, die folgende Funktionalitäten enthält:

- Eine Person hat die Lese-/Schreibeigenschaften *Name* und *Geschlecht*. Diese müssen immer gesetzt sein.
- Eine Person ist gleich einer anderen Person, wenn die Namen und das Geschlecht gleich sind. Mutter und Vater wird beim Testen auf Gleichheit nicht berücksichtigt.
- Zu jeder Person kann ihre *Mutter* und *Vater* – die wiederum Personen sind – bekannt sein.
- Eine Person enthält die Liste Ihrer *Kinder*. Diese Eigenschaft kann nur gelesen werden.
- Die Person stellt zudem ihre *Töchter*, *Söhne*, *Schwestern*, *Brüder* und *Nachkommen* bereit.

Gehen Sie beim Entwickeln der Klasse in folgenden Schritten vor. Wenden Sie dabei im Labor das *Pair Programming*-Konzept von XP an. Setzen Sie zuhause Ihre Programmtätigkeit alleine fort:

1. Erstellen Sie zuerst für die mitgelieferte Klasse `Person` die dazugehörige Testklasse `PersonTest`, und starten Sie den JUnit-Test.
2. Schreiben Sie zuerst Ihren ersten Test, indem Sie die Testmethode mit dem Namen `creation()` programmieren, welche über einen Custom-Konstruktor ein Personenobjekt anlegt. Dem Konstruktor werden Name und Geschlecht übergeben (**HINWEIS:** `Person p = new Person("Sepp", Gender.MALE);`). Der Test soll ebenfalls kontrollieren, ob die Eigenschaften korrekt gesetzt wurden (**HINWEIS:** `assertEquals()`).

   Um den Compiler-Fehler aus der Klasse `PersonTest` zu entfernen, ergänzen Sie anschließend die Klasse `Person` durch den entsprechenden Konstruktor. Starten Sie den JUnit-Test.
3. Da es keine Personen ohne Name (`name == null || name.length() == 0`) und Geschlecht (`gender == null`) geben darf, sollen Sie die nächsten Tests mit dem Namen `nameGenderEmpty()`, `nameEmpty()` und `genderEmpty()` schreiben, welche die Setter-Methoden aufrufen und bei fehlenden Parameterwerten auf das Werfen eines `IllegalArgumentException` testet (**HINWEISE:** `p.setName(null); p.setName(""); p.setGender(null);`). Starten Sie den JUnit-Test.

   Ändern Sie dann die Klasse `Person`, so dass sie auf fehlende Eigenschaften reagiert und ein `IllegalArgumentException` wirft (**HINWEIS:** Wählen Sie beim Werfen von Exceptions aussagekräftige Fehlermeldungen). Starten Sie den JUnit-Test.
4. Schreiben Sie Tests mit dem Namen `creationNameGenderEmpty()`, `creationNameEmpty()` und `creationGenderEmpty()` die fehlende Eigenschaften im Konstruktoraufruf testen (**HINWEIS:** `Person(null, null) Person("", null) Person("Sepp", null)`). Starten Sie den JUnit-Test.

   Ändern Sie dann die Klasse `Person`, so dass sie auf fehlende Eigenschaften im Konstruktor reagiert und ein `IllegalArgumentException` wirft. Starten Sie den JUnit-Test.
5. Schreiben Sie dann einen Test mit dem Namen `equals()` und einen mit dem Namen `equalsNull()`, die kontrollieren ob zwei Personen gleich sind. Zwei Personen sind gleich, wenn Name und Geschlecht gleich sind. Der Vergleich mit einem Objekt das keine Person ist, soll `false` liefern. Der Vergleich einer Person mit `null` soll ein `IllegalArgumentException` hervorrufen, Starten Sie den JUnit-Test.

   Ändern Sie dann die Klasse `Person` entsprechend ab, und starten Sie den JUnit-Test.
6. Schreiben Sie einen weiteren Test mit dem Namen `parent()`, der testet ob Mutter und Vater richtig gesetzt werden und insbesondere ob bei der Übergabe von `null` Mutter oder Vater gelöscht werden (**WICHTIG:** Testen Sie noch nicht die Richtigkeit des Geschlechts bei Mutter und Vater. Dies soll der nächste Test erledigen). Starten Sie den JUnit-Test.

   Ändern Sie die Methoden `setMother()` und `setFather()` entsprechend ab, und starten Sie den JUnit-Test.

7. Dieser nächste Test mit dem Namen `parentIncorrectGender()` kontrolliert beim Setzen oder Ändern von Mutter und Vater, ob diese das passende Geschlecht haben. Stimmt das Geschlecht nicht überein, wird wiederum ein `IllegalArgumentException` geworden. Starten Sie den JUnit-Test.

   Ändern Sie die Methoden `setMother()` und `setFather()` ab, und starten Sie den JUnit-Test.

8. Dieser Test mit dem Namen `children()` soll kontrollieren, ob beim Setzen von Mutter und Vater in einer Person auch die Kinder in ihrer Mutter und Vater richtig gesetzt werden. Dabei muss darauf geachtet werden, dass Kinder bei ihren Eltern nicht mehrmals eingetragen werden. Auch muss getestet werden, dass beim Löschen von Mutter und Vater in einer Person die Person als Kind bei ihren Eltern entfernt wird. Beim Ändern von Mutter und Vater muss das Kind bei einem Elternteil entfernt und beim anderen hinzugefügt werden (**HINWEIS:** `children` wurde deshalb vom Typ `ArrayList` gewählt, weil dadurch sehr einfach kontrolliert werden kann, wie viele Elemente `children` enthält, ob ein Element enthalten ist und an welcher Stelle das gesuchte Element in der Liste zu finden ist). Starten Sie den JUnit-Test.

   Ändern Sie die Methoden `setMother()` und `setFather()` entsprechend ab, und starten Sie den JUnit-Test.

9. Schreiben Sie einen weiteren Test mit dem Namen `daughtersSons()`, welcher die zu codierenden Methoden `getDaughters()` und `getSons()` testet. Gibt es keine Töchter oder Söhne, müssen die Methoden instanziierte `ArrayList`-Objekte die keine Elemente enthalten zurück liefern. Starten Sie den JUnit-Test.

   Codieren Sie dann die beiden Methoden, und starten Sie wiederum den JUnit-Test.

10. Der nächste Test mit dem Namen `sistersBrothers()` soll testen, ob die richtigen Schwestern und Brüder für eine Person geliefert werden können. Dabei gilt dass beim Fehlen nur eines einzigen Elternteils die Methoden `ArrayList`-Objekte mit keinen Elementen zurück liefern sollen. Halbbrüder und -schwestern dürfen nicht zurückgeliefert werden. Starten Sie den JUnit-Test.

    Programmieren Sie die Methoden `getSisters()` und `getBrothers()` aus. Starten Sie den JUnit-Test.

11. Jetzt sollen Sie einen Test schreiben der kontrolliert ob zu einer Person alle ihre Nachkommen geliefert werden. Der Test soll den Namen `descendants()` haben. Nachkommen dürfen in der Ergebnisliste nicht mehrmals vorkommen. Starten Sie den JUnit-Test.

    Programmieren Sie dann die Methode `getDescendants()`, und testen Sie diese mit JUnit-Test.

12. Zum Schluss sollen Sie Tests mit dem Namen `parentOfHimself()` und `parentIsDescendant()` schreiben, die beim Setzen eines Elternteiles darauf testen, dass eine Person nicht Elternteil von sich selbst ist und dass ein Elternteil nicht Nachkomme der Person sein darf. In beiden Fällen wird ein `IllegalArgumentException` geworfen. Starten Sie den JUnit-Test.

    Ändern Sie dann die Methoden `setMother()` und `setFather()` entsprechend ab, und testen Sie wiederum mit JUnit.

Nachdem Sie Ihre Klassen geschrieben und ausgiebig getestet haben, sollen Sie Ihre Testklasse mit jener Ihres Kollegen austauschen und Ihre Klasse `Person` durch diese testen. Diskutieren Sie mit Ihrem Kollegen die evtl. aufgetretenen Fehler, und bessern Sie diese bei Bedarf aus.

Im mitgelieferten Dokument mit dem Titel *Lesen aus Dateien beim Testen mit JUnit* wird Ihnen erklärt wie das Simulieren von Lesevorgängen aus Dateien beim Testen bewerkstelligt werden kann.

Sie sollen die mitgelieferte Klasse `PersonList` – welche von `ArrayList` abgeleitet ist – erweitern, so dass diese bei der Instanziierung Personen mit ihren Verwandtschaftsverhältnissen aus einer Datei aufnehmen kann. Berücksichtigen Sie dabei die vorgestellten Möglichkeiten, um für Testzwecke Dateiinhalte schnell aus Strings zu generieren. Gehen Sie dabei folgendermaßen vor:

1. Erstellen Sie zur Klasse `PersonList` die Testklasse `PersonListTest`. Legen Sie zuerst keine Testmethoden fest, und starten Sie den JUnit-Test.

2. Erstellen Sie dann für alle nachfolgenden Dateiinhalte einen eigenen Test, der erfolgreich ist, wenn das entsprechende `IllegalArgumentException` geworfen wird (**HINWEIS:** Achten Sie beim Werfen des Exceptions darauf, dass aus der Fehlermeldung ersichtlich ist, welche Person Probleme beim Importieren hervorgerufen hat).

   Nachdem ein Test erstellt wurde, sollen Sie die Methode `readPersons()` so anpassen, dass der Test erfolgreich bestanden wird. Erst dann sollen Sie den nächsten Test erstellen.

```
String fileString = "";
String fileString = "\n";
String fileString = "blabla";
String fileString = "blabla,blabla";
String fileString = "blabla;blabla";
String fileString = "blabla;blabla;blabla";
String fileString = "blabla;blabla;blabla;blabla;blabla";
String fileString = ";;;";
String fileString = "null;null;null;null";
String fileString = ";;null;null";
String fileString = "Sepp;;null;null";
String fileString = "Sepp;blabla;null;null";
String fileString = "Sepp;MALE;null;null\nblabla";
String fileString = "Sepp;MALE;null;null\nblabla,blabla";
String fileString = "Sepp;MALE;null;null\nblabla;blabla";
String fileString = "Sepp;MALE;null;null\nblabla;blabla;blabla";
String fileString = "Sepp;MALE;null;null\nblabla;blabla;blabla;blabla;blabla";
String fileString = "Sepp;MALE;null;null\n;;;";
String fileString = "Sepp;MALE;null;null\nnull;null;null;null";
String fileString = "Sepp;MALE;null;null\n;;null;null";
String fileString = "Sepp;MALE;null;null\nElsa;;null;null";
String fileString = "Sepp;MALE;null;null\nElsa;blabla;null;null";
String fileString = "Sepp;MALE;;";
String fileString = "Sepp;MALE;null;null\nSepp;MALE;null;null";
String fileString = "Sepp;MALE;Resi;null";
String fileString = "Sepp;MALE;null;Hugo";
```

3. Schreiben Sie weiters einen Test mit dem Namen `readPersonsCorrect()` welche testet, ob folgender String erfolgreich und richtig in `PersonList` aufgenommen wird:

```
String fileString =
   "Sepp;MALE;null;null\nRosi;FEMALE;null;null\n" +
   "Rudi;MALE;Rosi;Sepp\nElsa;FEMALE;Rosi;Sepp\nEdit;FEMALE;Rosi;Sepp\n" +
   "Hugo;MALE;Elsa;Rudi\nHerta;FEMALE;Elsa;Rudi";
```

   Passen Sie bei Bedarf die Methode `readPersons()` an, und testen Sie mit JUnit-Test.

4. Zum Schluss sollen Sie noch den Test `readPersonsFromFile()` schreiben, der testet ob das Lesen aus der mitgelieferten Datei `relationship.txt` richtig funktioniert.

   Passen Sie bei Bedarf die Methode `readPersons()` an, und testen Sie mit JUnit-Test.