README text file

##################
###Project name###
##################
codetest

##################
How to run
##################
*NOTE: All source input files are located at the project base directory - here is where both the main() method of the application and the JUnit tests look for them.
Additionally, screen prints of the application output and screen shots on building/running the application are located in the 'docs' directory in the zip file.

Unzip file the zip file codetest.zip and 'cd' (i.e., change directory) into codetest directory created using a Windows/Unix console. At the command prompt/console type the following: 'ant run' (assumes you have Ant configured/installed).

Alternately, you can unzip the file into your Integrated Development Environment workspace and run the main class or the JUnit tests from there.

**IMPORTANT – Regarding Program Output
The sort-order of the following 3 records in Output 2 may differ from the contents of the 'model_output.txt' file:
Bonk Radek Male 6/3/1975 Green
Bouillon Francis Male 6/3/1975 Blue
Kournikova Anna Female 6/3/1975 Red
The reason for this is due to the fact these 3 are sorted by 'Date ascending' as per the requirements. Since they all have the same Date, the 3 records can be sorted in any order and technically still satisfy the requirements (while not exactly matching the contents the 'model_output.txt' file).

##################
###How to build###
##################
Unzip file the zip file codetest.zip and 'cd' into codetest directory created using a Windows/Unix console.
Run the following command: ant
Ant will create jar file named codetest.jar in current directory.

####################
###Program design###
####################
*NOTE: Documentation exists also in the Comments of the Java source files.
There are 13 source files in 7 packages (not including the 2 Junit tests in the test.codetest.main package):

com.codetest.constants.DataFileConstants.java
- Common constants used by application.

com.codetest.constants.DelimiterDerivedRecordMetaData.java

- Each constant in this Enum represent the fields in each person data file.
  Additionally, each constant has properties that contains the 'Array Element Index' location for that particular field - for each type of delimited file.
  This is the constructor for the Enum - it makes it a little easier to picture:
  DelimiterDerivedRecordMetaData(int csvElementIndex, int spaceElementIndex, int pipeElementIndex)
  E.g., GENDER(2,3,3) indicates that the gender field exists in the 2nd array-element index in the comma file and at the 3rd array Element index in both the space and pipe files.

com.codetest.dao.PersonRecordDao.java
com.codetest.dao.PersonRecordDaoImpl.java

- This interface and accompanying implementation does two things:
  a) creates a PersonRecord from an actual record from the Person data file; and,
  b) creates a List of PersonRecords to return to the calling method.
  This getPersonRecord(String line, String filePrefix) method is unique in that it serves as both a dynamic DataMapper. The correct delimiter is identified by using the source filename. An Enum is used then to order the field-setting of each attribute and identify the correct element in the tokenized array that the current Enum corresponds.

com.codetest.domain.PersonRecord.java
com.codetest.domain.PersonRecordDataSource.java

- The PersonRecord domain object represents Person data
  We use only one PersonRecord object to represent both Persons with a middle initial and without because the only difference is the lack of a middle initial which implies absence of data and not a different PersonRecord per se. The PersonRecordDataSource is the class that represents the source files. If the person data records are held elsewhere besides a File then and object like this allows a little more flexibility with swapping out the persistence store.

com.codetest.driver.CodetestDriver.java

- The driver or Main-Class of our application.

com.codetest.comparator.DateComparator.java
com.codetest.comparator.GenderThenLastNameComparator.java
com.codetest.comparator.LastNameComparator.java

- Comparator implementations to allow the following sorting:
  a) by Gender (Female before Male), then Last Name ascending;
  b) by Date, ascending; and,
  c) by Last Name, descending.

com.codetest.exception.PersonRecordException.java

- Custom Exception for the application.

com.codetest.service.PersonRecordService.java
com.codetest.service.PersonRecordServiceImpl.java

- Service interface and implementation used to abstract data access and hold some business logic (for illustrative purposes, in our case the main() method holds more of

the data manipulation logic than the service).

##################
###JUnit tests###
##################

test.codetest.main.PersonRecordDaoTest.java

- Tests the parsing of the source person data input files and compares the returned PersonRecord domain object to the expected domain object.

test.codetest.main.PersonRecordListSortTest.java

- Tests both the retrieval of the List<PersonRecord> from the service method and also tests the expected 'sort-order' of the 3 sorting formats specified in the requirements.