CSD 3464 – ASSIGNMENT 03 (Question 01)

Overview:

The following question is **not** taken from *Absolute Java* (6th Ed.). Please follow the instructions included in **this** document and implement the following Java files:

⇒ EmployeeDemo.java *(Contains a `main()` method)*
⇒ Employee.java
⇒ Name.java (**Provided to you!**)

The above classes/files should all be inside a package called *q1*.

Perquisites:

Review the provided Name.java file. You will use this class to complete the below question.

Instructions:

Create an `Employee` class with a `static private` field that stores a positive `int numEmployees` that represents the total number of `Employee`s created. The class should also have the field `java.util.Calendar dateOfHire` to store the employees date of hire, `int empNum` to hold the unique employee number, and `Name name` to store the employee's name.

- `empNum` should be assigned inside **each** constructor based on the value of the `static` field `numEmployees`. The first `Employee` should have `empNum = 1`, second `Employee` should have `empNum = 2`, …
- Remember to increment the `static` field `numEmployees` by **one** inside each constructor. The `numEmployees` field should always properly maintain the number of `Employees` in your system at any given time.
- To set the `dateOfHire` value use the below code snippet:

```
dateOfHire = new GregorianCalendar(year, month, day); //sets Calendar object to specified year, month, day
```

The `Employee` class should have a two-parameter constructor which takes the two parameters `String name` and `String dateOfHire.` You should assume the `name` passed to the constructor is in the form "`First Last`" or "`First Middle Last`". Similarly, assume `dateOfHire` will be provided to the constructor in the format "`yyyy/mm/dd`". The constructor should set the `Employee` fields `dateOfHire`, `name`, and `empNum` to the appropriate values.

- To set the `name` field use the appropriate constructor from the provided `Name` class
    - If no middle name is provided, `name`'s `middle` field will be set to `null`

- To set `dateOfHire` use `StringTokenizer` **or** `String.split()`, along with the `Integer` wrapper class → `Integer.parseInt()` takes a `String` and converts it to its `int` equivalent

The `Employee` class should also have a copy constructor which takes `Employee empToCopy` and performs a deep copy of `empToCopy`. Remember a deep copy means you must also create a copy of the objects which compose `Employee` (i.e. `Name` and `Calendar`).

- Primitive values (`int`, `float`, etc.) and `String`s can simply be copied
- Values of type class must properly cloned
    - You can create a clone of `empToCopy`'s `dateOfHire` field by using `dateOfHire.clone()` which returns a clone of the invoked object
    - You can create a clone of `empToCopy`'s `name` field by using the `new` keyword to create a `new Name` object by using the class `Name`'s copy constructor provided

In terms of methods, the `Employee` class should have a `public static` method `getNumEmployees` which returns the `int` value currently stored in the `static int` field `numEmployees`.

In addition, the `Employee` class should have the `public` mutator methods `setDateOfHire, setDateOfHireDay, setDateOfHireMonth, setDateOfHireYear, setName, setFirstName, setMiddleName,` and `setLastName`.

- We have already shown how to set the entire date (year, month, and day), but the below code snippets allow you to set a specific element of a date

```
dateOfHire.set(Calendar.DATE, 3); //sets day
dateOfHire.set(Calendar.MONTH, 5); //sets month
dateOfHire.set(Calendar.YEAR, 2000); //sets year
```

- To update the `name` field, use the appropriate mutator method from the `Name` class

Furthermore, the `Employee` class should have the `public` accessor methods `getDateOfHire, getName, getFirstName, getMiddleName,` and `getLastName`.

- `getDateOfHire` returns the `String` representing the date the employee was hired. Use the below code snippet to format the returned `String`

```
SimpleDateFormat sdf = new SimpleDateFormat("MMMM dd, YY"); //specifies format for date
System.out.println(sdf.format(dateOfHire.getTime())); //converts date to specified format
```

- `getName` should simply return `name.toString()`
- `getMiddleName` should return "NONE" if middle name is `null`. Return middle name otherwise.

Your class must also include the method `public equals()` which compares two `Employee` objects and returns the `boolean` value `true` if and only if the `name`, `dateOfHire`, and `empNum` fields of both objects are the same.

- To compare primitive values (i.e. `empNum`) simply use `==`
- To compare the `dateOfHire` and `name` fields use their `equals()` methods already provided in both the `Calendar` and `Name` classes

Furthermore, the class `Money` must also include the method `public toString()` which returns a `String` object in the format "`empNum` : `name` - `dateOfHire`" where `dateOfHire` is in the format "month_name day, last_two_digits_year".

- Use `name.toString()` to get the `name` in the proper format
- Use the below code snippet to represent date in the correct format

```
SimpleDateFormat sdf = new SimpleDateFormat("MMMM dd, YY"); //specifies format for date
System.out.println(sdf.format(dateOfHire.getTime())); //converts date to specified format
```

You are also required to write a `static` class called `EmployeeDemo` in another file that contains a `main()`. The purpose of this class is to test the functionality of your `Employee` class. It is up to you to decided what to include in the `main()` to test the class; however, all `public` facing methods and constructors need to be tested for correctness.

The basic structure of your `Employee` class is required to look like the following:

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.GregorianCalendar;

public class Employee
{

     private static int numEmployees;

     private int empNum;
     private Name name;
     private Calendar dateOfHire;


     // Two Parameter Constructor
     public Employee(String name, String dateOfHire)
     {
         // insert code here
     }

     // Copy Constructor
     public Employee(Employee empToCopy)
     {
```

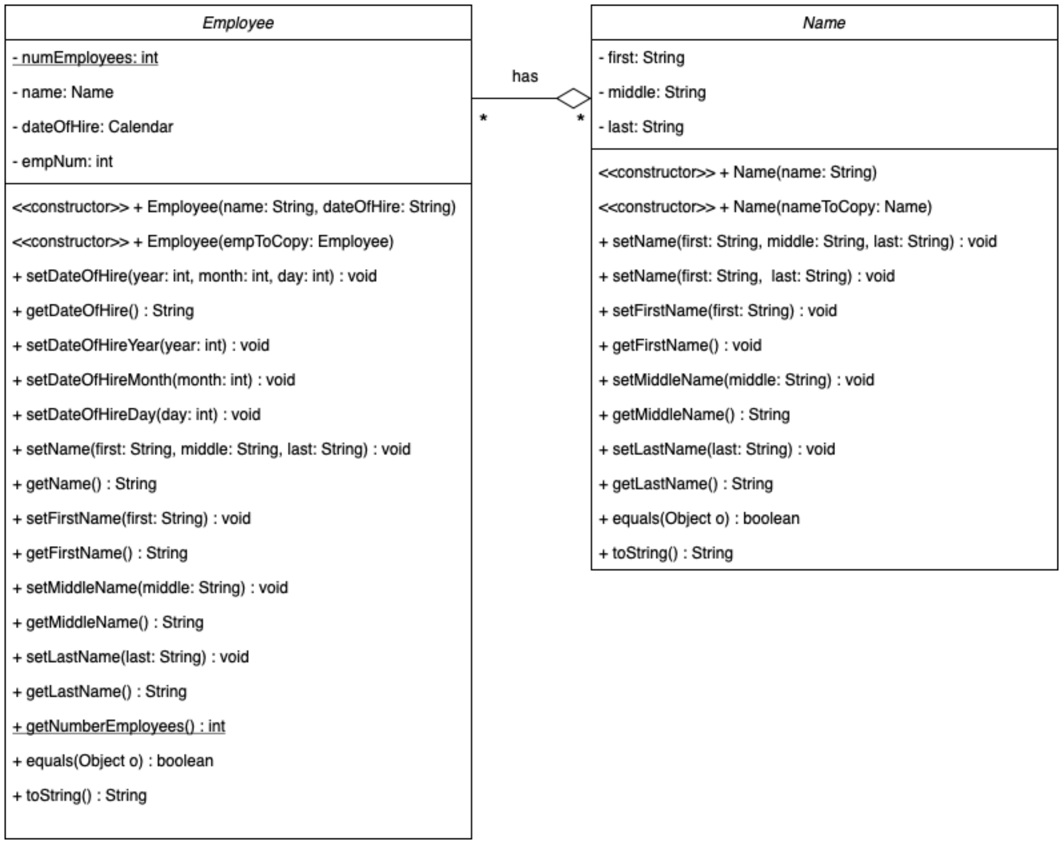
```
    // insert code here
}

// Mutator - DateOfHire
public void setDateOfHire(int year, int month, int day)
{
    // insert code here
}

// Accessor - DateOfHire
public String getDateOfHire()
{
    // insert code here
}


// Mutator - DateOfHireDay
public void set DateOfHireDay(int day)
{
   // insert code here
}

// Mutator - DateOfHireMonth
public void setDateOfHireMonth(int month)
{
   // insert code here
}

// Mutator - DateOfHireYear
public void setDateOfHireYear(int year)
{
   // insert code here
}

// Mutator - Name
public void setName(String first, String middle, String last)
{
   // insert code here
}

// Accessor – Name
public String getName()
{
    // insert code here
}

// Mutator - FirstName
public void setFirstName(String first)
{
   // insert code here
}
```

```
        // Accessor – FirstName
        public String getFirstName()
        {
            // insert code here
        }

        // Mutator - MiddleName
        public void setMiddleName(String middle)
        {
           // insert code here
        }

        // Accessor – MiddleName
        public String getMiddleName()
        {
            // insert code here
        }

        // Mutator - LastName
        public void setLastName(String last)
        {
           // insert code here
        }

        // Accessor – LastName
        public String getLastName()
        {
            // insert code here
        }

        // toString method
        @Override
        public String toString()
        {
           // insert code here
        }

        // equals method
        @Override
        public boolean equals(Object o)
        {
           // insert code here
        }

}
```

UML Diagram:

The below UML diagram is provided for your convivence. Ensure all shown fields, constructors, and methods are included in your Employee class prior to your final submission.

Note: <u>underlined</u> values denote `static` fields and methods.


Employee
- numEmployees: int
- name: Name
- dateOfHire: Calendar
- empNum: int
<<constructor>> + Employee(name: String, dateOfHire: String)
<<constructor>> + Employee(empToCopy: Employee)
+ setDateOfHire(year: int, month: int, day: int) : void
+ getDateOfHire() : String
+ setDateOfHireYear(year: int) : void
+ setDateOfHireMonth(month: int) : void
+ setDateOfHireDay(day: int) : void
+ setName(first: String, middle: String, last: String) : void
+ getName() : String
+ setFirstName(first: String) : void
+ getFirstName() : String
+ setMiddleName(middle: String) : void
+ getMiddleName() : String
+ setLastName(last: String) : void
+ getLastName() : String
+ getNumberEmployees() : int
+ equals(Object o) : boolean
+ toString() : String
has
*
*
Name
- first: String
- middle: String
- last: String
<<constructor>> + Name(name: String)
<<constructor>> + Name(nameToCopy: Name)
+ setName(first: String, middle: String, last: String) : void
+ setName(first: String, last: String) : void
+ setFirstName(first: String) : void
+ getFirstName() : void
+ setMiddleName(middle: String) : void
+ getMiddleName() : String
+ setLastName(last: String) : void
+ getLastName() : String
+ equals(Object o) : boolean
+ toString() : String