Assignment 2: to be done at home, with no partner

Contents

[Assignment 2 of 3: to be done at home on your own (no partner) 2](#_Toc357174263)

[General Mark Deductions: 2](#_Toc357174264)

[Class requirement details for variables, constructors, methods, and comments. 3](#_Toc357174265)

[Date class’s public String getDayOfWeek() method algorithm: 10](#_Toc357174266)

[TestDate class 13](#_Toc357174267)

[TestAssignment Class and MARKING GUIDE 14](#_Toc357174268)

[Expected output of TestAssignment constructor: 22](#_Toc357174269)

[Submission Due Date and Requirements 27](#_Toc357174270)

**If your class does not compile, you will receive a mark of 0.**

**This is an individual assignment. Group work on this will be considered plagiarism!**

# Assignment 2 of 3: to be done at home on your own (no partner)

Assignment 2 involves writing multiple classes, as described below. **Lates not accepted.**

## General Mark Deductions:

|  |  |
| --- | --- |
| Requirement | Deductions if not done |
| Create a project named Assignment2 | -1 |
| Bad indentation or difficult-to-read/follow code | -1 each |
| Each public method must have a javadoc comment with an @param or @return tag (as appropriate) | -1 each |
| Your code has magic numbers. | -1 each |
| Zip your file and named it as follows:  lastname-firstname-assignment-2-comp-1409.zip (use your own last name and first name: e.g. woods-tiger-assignment-2-comp-1409.zip)  …and upload it to BCIT’s server before the due date (see last page). | -2 |
| Total mark | /102 |

## Class requirement details for variables, constructors, methods, and comments.

This assignment consists of six classes you will make:

* Date
* Name
* Director
* Movie
* TestDate
* TestAssignment

|  |  |  |  |
| --- | --- | --- | --- |
| **class** | **Instance variables with accessors and mutators including javadocs and @param tags and @return tags as appropriate for each. Mutators have same rules as constructors for data validation.** | **Constructors with @param tags and/\*\* javadoc \*/comments.** | **Other data and methods with including /\*\* javadocs \*/ and @param tags and @return tags as appropriate for each public method and /\* normal comments \*/ for private methods.** |
| **Date** | **Year** e.g. 2007  **Month** e.g. 12  **Day** e.g. 31 | 1. **public Date()** Sets date to January 1 2017 2. **public Date(int year, int month, int day)** ensures dates are valid (e.g. February 29 2012 is ok but February 29 2013 is not). When invalid dates are entered, reset the date to January 1, 2017 instead. The only valid years are 0 to 2017. | 1. Symbolic constants for each of the **twelve months (ints and Strings)** and **days of the week**   e.g. JANUARY = 1  e.g. FEBRUARY = 2  etc  e.g. SATURDAY = 0  e.g. SUNDAY = 1  etc  e.g. JANUARY\_STRING = "January"  e.g. FEBRUARY\_STRING = "February"  etc  CURRENT\_YEAR = 2017   1. **public String getDayOfTheWeek()** returns the day of the week. [See algorithm below](#_Date_class’s_public) on later pages. 2. **private Boolean isLeapYear()…**true for every year divisible by 400. Also true for every year divisible by 4 UNLESS it is also divisible by 100 but not 400. 3. **public int getNumberOfDaysInThisMonth()** returns the number of days in the instance-variable month 4. **public String getFormattedDate()** returns a string in the following format: yyyy-mm-dd. For example:   February 29, 2004: 2004-02-29  December 1, 1999: 1999-12-01  etc   1. **private String getPaddedMonth()** which returns a string in the following format:   Month 11: returns "11"  Month 9: returns "09"  Month 1: returns "01"  Etc   1. **private String getPaddedDay()** which returns a string in the following format:   Day 31: returns "31"  Day 20: returns "20"  Day 2: returns "02"  Day 7: returns "07"  etc   1. **public static String getMonthName(int monthNumber)**   e.g. returns "January" for 1  e.g. returns "November" for 11  returns null for monthNumber values not between 1 and 12  etc   1. **public static int getMonthNumber(String monthName)**   e.g. returns 4 for "April"  e.g. returns 4 for "apRiL"  e.g. returns 5 for "MAY"  etc  returns 0 for unknown months |
| **Name** | String first  String last  String middle | 1. **public Name()** sets name to "A. Non Ymous" for first, middle, last. 2. **public Name(String first, String last, String middle)** ensures first and last are not null and also that all three names are between 1-40 characters in length. NOTE: it is ok for the middle name to be null. If any of the names is too short or too long, set the name to "A. Non Ymous" for first, middle, last; also, if the first or last names are null, set the name to "A. Non Ymous" for first, middle, last. | 1. symbolic constants for MINIMUM\_NAME\_LENGTH = 1 and MAXIMUM\_NAME\_LENGTH = 40 2. **private String makePrettyName()** which returns the name in the following format:   Tiger Woods (no middle)  or  Tiger Tont Woods (middle)  …even if the names were stored in different cases such as:  tIGEr wooDs or tIGEr tONT wooDs   1. **public String getInitials()** returns strings in following format:   T.W. (no middle)  or  T.T.W. (middle)  …even if the names were stored in different cases such as:  tIGEr wooDs or tIGEr tONT wooDs   1. **public String getFullName()** returns the String created by makePrettyName()…by calling it |
| **Director** | Name name  Date born  Date died  Name pseudonym | 1. **public Director()** sets names and dates to new Date and Name objects by calling their default constructors 2. **public Director(String firstName, String lastName, String middleName, int yearBorn, int monthBorn, int dayBorn, int yearDied, int monthDied, int dayDied, String pseudonymFirstName, String pseudonymLastName, String pseudonymMiddleName)** creates Name and Date objects from the parameters by calling the appropriate Name and Date constructors. Don’t worry about invalid data being passed in; the only case we care about is if the pseudonym first, last, and middle names are ALL null, do not create a pseudonym Name object at all. 3. **public Director(Name name, Date born, Date died, Name pseudonym)** stores the supplied parameters. If the name or born parameter is null, call its class’s default constructor instead. 4. **public Director(Name name, Date born, Date died)** stores the supplied parameters and sets the pseudonym to null. If the name or born parameter is null, call its class’s default constructor instead. | 1. **public boolean isDirectorAlive()**   returns true if died is null; otherwise returns false   1. **public int getAgeYearsOfDirector() returns how old the director is (or would be…if they are dead): NOTE: do the simple arithmetic operation of “current year minus birth year” regardless of whether the director is dead or alive.** |
| **Movie** | Director director  Date released  String title | 1. **public Movie()** sets Director and Date to new Director and Date objects by calling their default constructors, and sets the title to "Untitled" 2. **public Movie(Director director, Date released, String title)** stores the supplied parameters. If the director is null, call its default constructor. If the date is null, call its default constructor. If the title is null, set it to "Untitled". | 1. **public String getDirectorName()** calls the director’s name’s getFullName() method 2. **public String getDayOfTheWeekMovieWasReleased()** calls the released Date object’s method to get the day of the week 3. **public void printDetails()** prints the Movie object’s details exactly in the following format:   Steven Allan Spielberg (S.A.S.) released Jaws on Friday, June 20, 1975, under the pseudonym Ralph Danger  or  Steven Allan Spielberg(S.A.S.) released Back to the Future on Friday, July 3, 1985  Or  Steven Spielberg(S.S.) released Jurassic World on Tuesday, October 20, 2015 |
| **TestDate** |  | All the functionality is in the default constructor | Details are [below](#_TestDate_class) |
| **TestAssignment** |  | All the functionality is in the default constructor | Details are [below](#_TestAssignment_Class) |

## Date class’s public String getDayOfWeek() method algorithm:

Add the following to your Date class. The method signature must be:

**public String getDayOfTheWeek()**

This method returns the day of the week (e.g. "Wednesday") for a specified date (e.g. October 31, 2012).

It must make use of a private method whose method signature is:

**private boolean isLeapYear()** (see: <http://en.wikipedia.org/wiki/Leap_year>)

Which returns true (e.g. for 1996, 2000, 2012, etc) or false (e.g. for 1900, 2011, etc) depending on whether a year is a leap year or not.

Here is the algorithm to determine what day of the week a given date is:

Example dates: **August 16, 1989** and **March 20, 1950**

Step 1: Only look at the last two digits of the year and determine how many 12s fit in it

**7** 12s in 89 **4** 12s in 50

Step 2: Look at the remainder of this division:

89 – 7 \* 12 = **5** 50 – 4 \* 12 = **2**

Step 3: How many 4s fit into that remainder:

**1** four in 5 **0** fours in 2

Step 4: Add the day of the month:

**16** for the 16th **20** for the 20th

Step 5: Add the month code:

3 for August **4** for March

|  |  |  |
| --- | --- | --- |
| Jan = 1 | Feb = 4 | Mar = 4 |
| Apr = 0 | May = 2 | Jun = 5 |
| Jul = 0 | Aug = 3 | Sep = 6 |
| Oct = 1 | Nov = 4 | Dec = 6 |

Step 6: Add your numbers, then mod by 7:

**7 + 5 + 1 + 16 + 3 = 32** 4 + 2 + 0 + 20 + 4 = 30  
 **32 % 7 = 4**  30 % 7 = 2

This is your day of the week, as follows:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Sat = 0 | Sun = 1 | Mon = 2 | Tue = 3 | **Wed = 4** | Thu = 5 | Fri = 6 |

**August 16, 1989** March 20, 1950  
 **Wednesday**  Monday

**NOTE**: some dates require special offsets:

January and February dates in leap years: subtract 1 from step 5

Dates in the 1600s: add 6 to step 5  
Dates in the 1700s: add 4 to step 5  
Dates in the 1800s: add 2 to step 5  
Dates in the 2000s: add 6 to step 5  
Dates in the 2100s: add 4 to step 5

## TestDate class

The TestDate class will help ensure whether your Date class methods are working. Create a class called **TestDate** with an array variable that stores Date’s named sampleDates[]. **(1 POINT)**

Create a constructor that makes fifteen new Date objects and then instantiates and stores those objects into sampleDates[]. **(4 POINTS)**

Create a method in TestDate class named getSampleDates() which uses a **while** loop to access the array and display the day of the week for the dates by calling each Date’s getDayOfTheWeek() method. The dates are specifically these ones, in this precise order: **(5 POINTS)**

November 15, 1970 Sunday

July 31, 1887 Sunday

May 2, 1966 Monday

August 19, 1980 Tuesday

September 11, 2001 Tuesday

June 26, 1900 Tuesday

February 28, 1940 Wednesday

October 30, 1974 Wednesday

January 15, 1914 Thursday

October 1, 1840 Thursday

December 31, 1999 Friday

May 20, 1988 Friday

March 10, 2012 Saturday

April 1, 2006 Saturday

February 29, 1992 Saturday

## TestAssignment Class and MARKING GUIDE

The TestAssignment class will help ensure whether your classes’ methods are working. Your instructor has already created a class called **TestAssignment** whose constructor does exactly what is described below in the same order too. If your assignment is working as expected, then *not one single character* will differ from what is shown here as output, and **there will be no Runtime Exceptions**. **Marks will stop once your assignment crashes**. This class is available for downloading on BCIT’s server: /OUT/COMP/1409/assignment2/TestAssignment.java

|  |  |  |
| --- | --- | --- |
| TestAssignment code | Output | Deductions if wrong/missing |
| public TestAssignment(){ |  |  |
| TestDate td = new TestDate(); | Sunday  Sunday  Monday  Tuesday  Tuesday  Tuesday  Wednesday  Wednesday  Thursday  Thursday  Friday  Friday  Saturday  Saturday  Saturday | 5 |
|  |  |  |
| Date d = new Date();  System.out.println(d.getFormattedDate());  // calls private method getPaddedMonth()  // calls private method getPaddedDay() | 2017-01-01 | 1  1  1 |
| System.out.println(Date.getMonthName(3)); | March | 1 |
| System.out.println(d.getDayOfTheWeek());  // calls private method isLeapYear() | Sunday | 1  1 |
| System.out.println(d.getNumberOfDaysInThisMonth ()); | 31 | 1 |
| System.out.println(Date.getMonthNumber("December")); | 12 | 1 |
|  |  |  |
| Date d2 = new Date(2001, 6, 29);  System.out.println(d2.getFormattedDate()); | 2001-06-29 | 1 |
| System.out.println(d2.getDayOfTheWeek()); | Friday | 1 |
| System.out.println(d2. getNumberOfDaysInThisMonth ()); | 30 | 1 |
|  |  |  |
| Date d3 = new Date(1995, 14, 8); // invalid date; month is > 12 System.out.println(d3.getFormattedDate()); | 2017-01-01 | 1 |
| System.out.println(d3.getDayOfTheWeek()); | Sunday | 1 |
| System.out.println(d3. getNumberOfDaysInThisMonth ()); | 31 | 1 |
|  |  |  |
| System.out.println(d3.getYear()); | 2017 | 1 |
| System.out.println(d3.getMonth()); | 1 | 1 |
| System.out.println(Date.getMonthName(d3.getMonth())); | January | 1 |
|  |  |  |
| d3.setYear(3500); // too high; don’t set it  System.out.println(d3.getYear()); | 2017 | 1 |
| d3.setYear(-3); // too low; don’t set it  System.out.println(d3.getYear()); | 2017 | 1 |
| d3.setYear(2010);  System.out.println(d3.getYear()); | 2010 | 1 |
| d3.setMonth(13); // too high; don’t set it  System.out.println(d3.getMonth()); | 1 | 1 |
| d3.setMonth(0); // too low ; don’t set it  System.out.println(d3.getMonth()); | 1 | 1 |
| d3.setMonth(8);  System.out.println(d3.getMonth()); | 8 | 1 |
|  |  |  |
| d3.setDay(32); // too high; don’t set it  System.out.println(d3.getDay()); | 1 | 1 |
| d3.setDay(0); // too low ; don’t set it  System.out.println(d3.getDay()); | 1 | 1 |
| d3.setDay(27);  System.out.println(d3.getDay()); | 27 | 1 |
|  |  |  |
| Name n = new Name();  System.out.println(n.getFullName());  // calls makePrettyName() | A. Non. Ymous. | 1  1 |
| System.out.println(n.getInitials()); | A.N.Y. | 1 |
| Name n2 = new Name("sTEven", "SpielberG", null);  System.out.println(n2.getFullName()); | Steven Spielberg | 1 |
| System.out.println(n2.getInitials()); | S.S. | 1 |
| Name nn = new Name("sTEven","SpielberG", "aLLan");  System.out.println(nn.getFullName()); | Steven Allan Spielberg | 1 |
| System.out.println(nn.getInitials()); | S.A.S. | 1 |
|  |  |  |
| Name n3 = new Name(null, "spielberg", "allan");  System.out.println(n3.getFullName()); | A. Non. Ymous. | 1 |
| System.out.println(n3.getInitials()); // first cannot be null | A.N.Y. | 1 |
|  |  |  |
| Name n4 = new Name("steVEn", null, "alLAn");  System.out.println(n4.getFullName()); // last cannot be null | A. Non. Ymous. | 1 |
| System.out.println(n4.getInitials()); | A.N.Y. | 1 |
|  |  |  |
| Name n5 = new Name("", "spielberg", "Allan"); // too short  System.out.println(n5.getFullName()); | A. Non. Ymous. | 1 |
| System.out.println(n5.getInitials()); | A.N.Y. | 1 |
|  |  |  |
| Name n6 = new Name(null, "spielberg", "alLAn"); // null first not allowed  System.out.println(n6.getFullName()); | A. Non. Ymous. | 1 |
| System.out.println(n6.getInitials()); | A.N.Y. | 1 |
|  |  |  |
| Name n7 = new Name("Steven", "spielberg", "01234567890123456789012345678901234567890123456789");  System.out.println(n7.getFullName()); // too long | A. Non. Ymous. | 1 |
| System.out.println(n7.getInitials()); | A.N.Y. | 1 |
|  |  |  |
| Name n8 = new Name("sTEven", "SpielberG", null);  System.out.println(n8.getFirstName()); | Steven | 1 |
| System.out.println(n8.getMiddleName()); | null | 1 |
| System.out.println(n8.getLastName()); | Spielberg | 1 |
|  |  |  |
| Name n9 = new Name("sTEven", "SpielberG", "alLAn");  System.out.println(n9.getMiddleName()); | Allan | 1 |
|  |  |  |
| Director a = new Director(); // default dates and names  System.out.println(a.getDateBorn().getFormattedDate());//default | 2017-01-01 | 1 |
| System.out.println(a.getName().getFullName()); //default | A. Non. Ymous. | 1 |
| System.out.println(a.getDateDied().getFormattedDate());//default | 2017-01-01 | 1 |
| System.out.println(a.getName().getFullName()); //default | A. Non. Ymous. | 1 |
| System.out.println(a.isDirectorAlive()); | false | 1 |
| System.out.println(a.getAgeYearsOfDirector()); // note: it is 2017 now | 0 | 1 |
|  |  |  |
| Date d1 = new Date(1946, 12, 18);  Director a1 = new Director(null, d1, null); // null name not allowed  System.out.println(a1.getName().getFullName()); | A. Non. Ymous. | 1 |
|  |  |  |
| Name n1 = new Name("sTEven", "SpielberG", null);  Director a2 = new Director(n1, null, null); // null born not allowed  System.out.println(a2.getDateBorn().getFormattedDate()); | 2017-01-01 | 1 |
|  |  |  |
| Director a3 = new Director(n1, d1, null);  System.out.println(a3.getDateBorn().getFormattedDate()); | 1946-12-18 | 1 |
| System.out.println(a3.getName().getFullName()); | Steven Spielberg | 1 |
|  |  |  |
| Name name = new Name("sTEven", "SpielberG", "alLAn");  Date born = new Date(1946, 12, 18);  Name pseudonym = new Name("RalpH", "DanGEr", null);  Director a4 = new Director(name, born, null, pseudonym);  System.out.println(a4.getName().getFullName()); | Steven Allan Spielberg | 1 |
| System.out.println(a4.getDateDied()); | null | 1 |
| System.out.println(a4.getDateBorn().getFormattedDate()); | 1946-12-18 | 1 |
| System.out.println(a4.getPseudonym().getFullName()); | Ralph Danger | 1 |
|  |  |  |
| Name name1 = new Name("sTEven", "SpielberG", "alLAn");  Date born1 = new Date(1946, 12, 18);  Director a5 = new Director(name1, born1, null); // no pseudonym  System.out.println(a5.getName().getFullName()); | Steven Allan Spielberg | 1 |
| System.out.println(a5.getDateDied()); | null | 1 |
| System.out.println(a5.getDateBorn().getFormattedDate()); | 1946-12-18 | 1 |
| System.out.println(a5.getPseudonym()); | null | 1 |
|  |  |  |
| Name name2 = new Name("sTEven", "SpielberG", "alLAn");  Date born2 = new Date(1946, 12, 18);  Date died = new Date(2000, 6, 13);  Director a6 = new Director(name2, born2, died);  System.out.println(a6.getDateDied().getFormattedDate()); | 2000-06-13 | 1 |
|  |  |  |
| Director a7 = new Director("steven", "spielberg", null, 1946, 12, 18, 2000, 6, 13, "ralph", "danger", null);  System.out.println(a7.getDateDied().getFormattedDate()); | 2000-06-13 | 1 |
| System.out.println(a7.getName().getFullName()); | Steven Spielberg | 1 |
| System.out.println(a7.getDateBorn().getFormattedDate()); | 1946-12-18 | 1 |
| System.out.println(a7.getPseudonym().getFullName()); | Ralph Danger | 1 |
|  |  |  |
| Name name3 = new Name("sTEven", "SpielberG", "alLAn");  Date born3 = new Date(1946, 12, 18);  Date died3 = new Date(2017, 2, 1);  Director a8 = new Director(name3, born3, died3);  System.out.println(a8.getAgeYearsOfDirector()); // note: it’s 2017 now | 71 | 1 |
|  |  |  |
| Movie bb = new Movie(); // default title is "Untitled"  System.out.println(bb.getTitle()); | Untitled | 1 |
| System.out.println(bb.getDirector().getName().getFullName()); | A. Non. Ymous. | 1 |
| System.out.println(bb.getDirector().getPseudonym().getFullName()); | A. Non. Ymous. | 1 |
| System.out.println(bb.getDirector().getDateBorn().getFormattedDate()); | 2017-01-01 | 1 |
| System.out.println(bb.getDirector().getDateDied().getFormattedDate()); | 2017-01-01 | 1 |
| System.out.println(bb.getDateReleased().getFormattedDate()); | 2017-01-01 | 1 |
|  |  |  |
| Date date = new Date(1985, 7, 3);  String title = "Back to the Future";  Movie b2 = new Movie(null, date, title); // Director is null  System.out.println(b2.getDirector().getName().getFullName()); | A. Non. Ymous. | 1 |
| System.out.println(b2.getTitle ()); | Back to the Future | 1 |
| System.out.println(b2. getDateReleased().getFormattedDate()); | 1985-07-03 | 1 |
|  |  |  |
| Movie b3 = new Movie(null, null, null); // Date, title are null  System.out.println(b3. getDateReleased().getFormattedDate()); | 2017-01-01 | 1 |
| System.out.println(b3.getTitle ()); | Untitled | 1 |
|  |  |  |
| Name name4 = new Name("sTEven", "SpielberG", "alLan");  Date born4 = new Date(1946, 12, 18);  Date died4 = new Date(2000, 6, 13);  Director aa = new Director(name4, born4, died4);  Date released4 = new Date(2015, 10, 20);  Movie b4 = new Movie(aa, released4, "Jurassic World");  System.out.println(b4.getDirectorName ()); | Steven Allan Spielberg | 1 |
| System.out.println(b4.getDayOfTheWeekMovieWasReleased()); | Tuesday | 1 |
| b4.printDetails(); | Steven Allan Spielberg (S.A.S.) released Jurassic World on Tuesday, October 20, 2015 | 1 |
| Name name5 = new Name("sTEven", "SpielberG", null);  Date born5 = new Date(1946, 12, 18);  Date died5 = new Date(2000, 6, 13);  Name pseudonym5 = new Name("Ralph", "Danger", null);  Director aaa = new Director(name5, born5, died5, pseudonym5);  Date released5 = new Date(2015, 10, 20);  Movie b5 = new Movie(aaa, released5, "Jurassic World");  b5.printDetails();  // use the getDayOfTheWeekMovieWasReleased() method | Steven Spielberg (S.S.) released Jurassic World on Tuesday, October 20, 2015 , under the pseudonym Ralph Danger | 1  1 |
|  |  |  |
| Total |  | 102 |

## Expected output of TestAssignment constructor:

Sunday

Sunday

Monday

Tuesday

Tuesday

Tuesday

Wednesday

Wednesday

Thursday

Thursday

Friday

Friday

Saturday

Saturday

Saturday

2017-01-01

March

Sunday

31

12

2001-06-29

Friday

30

2017-01-01

Sunday

31

2017

1

January

2017

2017

2010

1

1

8

1

1

27

A. Non Ymous

A.N.Y.

Steven Spielberg

S.S.

Steven Allan Spielberg

S.A.S.

A. Non Ymous

A.N.Y.

A. Non Ymous

A.N.Y.

A. Non Ymous

A.N.Y.

A. Non Ymous

A.N.Y.

A. Non Ymous

A.N.Y.

Steven

null

Spielberg

Allan

2017-01-01

A. Non Ymous

2017-01-01

A. Non Ymous

false

0

A. Non Ymous

2017-01-01

1946-12-18

Steven Spielberg

Steven Allan Spielberg

null

1946-12-18

Ralph Danger

Steven Allan Spielberg

null

1946-12-18

null

2000-06-13

2000-06-13

Steven Spielberg

1946-12-18

Ralph Danger

71

Untitled

A. Non Ymous

A. Non Ymous

2017-01-01

2017-01-01

2017-01-01

A. Non Ymous

Back to the Future

1985-07-03

2017-01-01

Untitled

Steven Allan Spielberg

Tuesday

Steven Allan Spielberg(S.A.S.) released Jurassic World on Tuesday, October 20, 2015

Steven Spielberg(S.S.) released Jurassic World on Tuesday, October 20, 2015, under the pseudonym Ralph Danger

## Submission Due Date and Requirements

**Final note: make sure you have accessor and mutator methods for every instance variable in every class! Many students omit these.**

TO **SUBMIT THIS ASSIGNMENT** : Upload it to the learn.bcit.ca drop box for the Assignment 2.

**Late assignments are not accepted.**

**This assignment is due before 11:59pm the night before lesson 9.**