readme

/* Checks whether the given year is a leap year.   * @return true if and only if the input year is a leap year.   */  public static boolean isLeapYear(int year) {    ...  }

/** Returns the number of days in a given month.   * @param month is a month, numbered in the range 1...12.   * @param year is the year in question, with no digits omitted.   * @return the number of days in the given month.   */  public static int daysInMonth(int month, int year) {    ...  }

/** Checks whether the given date is valid.   * @return true if and only if month/day/year constitute a valid date.   * @return false if the day is out of range, the month out of range,   * or the year is negative.   */  public static boolean isValidDate(int month, int day, int year) {    ...  }

Part II

Define the internal state that a "Date" object needs to have by declaring some data fields (all private) within the Date class. Define the basic constructor specified below. A Date should be constructed only if the date is valid. If a caller attempts to construct an invalid date, the program should halt after printing an error message of your choosing. To halt the program, include the line:

System.exit(0);

/** Constructs a date with the given month, day and year.   * If the date is not valid, the entire program will halt with an error message.   * @param month is a month, numbered in the range 1...12.   * @param day is between 1 and the number of days in the given month.   * @param year is the year in question, with no digits omitted.   */  public Date(int month, int day, int year) {    ...  }

/** Returns a string representation of this date in the form month/day/year.   * The month, day, and year are expressed in full as integers; for example,   * 12/7/2006 or 3/21/407.   * @return a String representation of this date.   */  public String toString() {    ...  }

The file Date.java contains a skeleton, plus some test code, for a Date class. Your job is to fill in the implementations of the methods. We have specified most or all of the methods you’ll need, including some helper methods.

* The Unix "cal" command will remind you of the number of days in each month. February contains 28 days most years, but 29 days during a leap year. A leap year is any year divisible by 4, except that a year divisible by 100 is not a leap year, except that a year divisible by 400 is a leap year after all. Hence, 1800 and 1900 are not leap years, but 1600 and 2000 are. (Implement this rule in your program even if you know information to the contrary.)
readme

Part III
--------
Implement the following methods.

/** Determines whether this Date is before the Date d. @return true if and only if this Date is before d. */
public boolean isBefore(Date d) {
  ...
}

/** Determines whether this Date is after the Date d. @return true if and only if this Date is after d. */
public boolean isAfter(Date d) {
  ...
}

/** Returns the number of this Date in the year. @return a number n in the range 1...366, inclusive, such that this Date is the nth day of its year. (366 is used only for December 31 in a leap year.) */
public int dayOfYear() {
  ...
}

/** Determines the difference in days between d and this Date. For example, if this Date is 12/15/2012 and d is 12/14/2012, the difference is 1. If this Date occurs before d, the result is negative. @return the difference in days between d and this date. */
public int difference(Date d) {
  ...
}

---

Part IV
--------
Implement the final missing piece of your class, a second constructor that takes a String argument.

/** Constructs a Date object corresponding to the given string. @param s should be a string of the form "month/day/year" where month must be one or two digits, day must be one or two digits, and year must be between 1 and 4 digits. If s does not match these requirements or is not a valid date, the program halts with an error message of your choice. */
public Date(String s) {
  ...
}

We’re flexible on how you handle dates that are "almost correct". For example, the string "11/4/2010 AD" is technically not valid because of the spaces and letters, but it’s your choice whether you treat it the same as "11/4/2010" or halt with an error message. We don’t want to be pedantic about this. But your Date constructor definitely should not accept "11/31/2009" or "12/4" or "hey dude".

Hint: use the online Java API to familiarize yourself with all the methods available to you in the String class.

Submitting your solution
--------------------------
Change (cd) to your hw2 directory, which should contain Date.java. Make sure your code compiles and your tests run correctly on the _lab_ machines just before you submit it. From your hw2 directory, type "submit hw2".

WARNING: make sure your code _compiles_ and _runs_ on the _lab_machines_ right before you submit it. Every semester, we get dozens of homework submissions that don’t even compile. Don’t make a tiny last-minute change and assume your code still compiles. Don’t assume because the code works on your laptop that it will work in the lab. You will not receive sympathy for code that "almost" works.

After submitting, if you realize your solution is flawed, you may fix it and submit again. You may submit as often as you like. Only the last version you submit before the deadline will be graded.