```
package Lab12;

public class Book {
	public int yearPub;	//ano de publicação do livro
	public String title;	//título
	public String publisher;
	int requestNum = 0;	//número de requisições do livro
	boolean bookIn= true; // é possivel requisitar o livro
	private Date lastRequest;	//data da última requisição do livro

	/**
	 * The Book constructor
	 * @param title
	 * @param publisher
	 */
	public Book(String title, String publisher){	//construtor
		this.title = title;
		this.publisher = publisher;
	}

	/**
	 * Return book's title
	 * @param title
	 * @return
	 */
	public String getTitle(){ // Retorna o título do livro.
		return title;
	}

	/**
	 * Return Book's publisher
	 * @return
	 */
	public String getPublisher() {//Retorna a editora do livro.
		return publisher;
    }

	/**
	 * Return the Book's requisition date
	 * @return
	 */
	public Date getRequestDate(){ //Retorna a data da requisição do livro.
	   return lastRequest;
	}

	/**
	 * Return the date of the Book's last requisition
	 * @return
	 */
	public Date getLastRequestDate(){ //Retorna a data da última requisição do livro.
	   return lastRequest;
	}

	/**
	 * Return the total number of Book's requests
	 * @return
	 */
	public int getTotalRequests(){ //Retorna o numero total de requisições do livro.
```

```
            return requestNum;
        }

        /**
         * Order the book.
         * @return if possible returns true, otherwise returns false
         */
        boolean requestIt(Date today){  //Requisitar o livro, se for possível retorna
true, caso contrário, retorna false.
            if (bookIn){
                bookIn = false;
                requestNum ++;
                lastRequest = today;
                return true;
            }
            else
                return false;
        }

        /**
         * Returns the book,
         * @return if possible returns true, otherwise returns false
         */
        public boolean returnIt(){  //Retorna o livro, se for possível retorna true, caso
contrário, retorna false.
            if (bookIn)
                return false;
            else
            {
                bookIn = true;  //Se o livro está disponível para ser requisitado
                return true;
                }
        }

        /**
         * Print the book's information
         */
        public void print(){   //Mostra na consola a informação do livro.
                System.out.println("Book's information");
                System.out.println("Title: " + getTitle());
                System.out.println("publisher: " + getPublisher());
                System.out.println("Book is avaiable: " + bookIn);
        }
}
```

```
package Lab12;

public class Date {
        private int day;
        private int month;
        private int year;

/**
 * The constructor have the following parameters
 * @param day
 * @param month
 * @param year
```

```
 * checks if the numbers introduced are according with the calendar
 * if the values introduced are out of  boundaries return a message "invalid value"
 */
      public Date(int day , int month, int year){   //construtor - verificação se a data foi introduzida corretamente
          this.day = day;
          this.month = month;
          this.year = year;

          if (year > 0){
              if ((month > 0)&&(month <= 12))
              {
                 DaysInTheMonth(year,month);
                 if (day>DaysInTheMonth(year,month)) // se o dia introduzido for superior ao número de dias do mês
                 {
                     throw new IllegalArgumentException();  // lançar excepção
                 }
              }
              else  // se o mês introduzido é inferior a 1 ou superior a 12.
              {
                 throw new IllegalArgumentException();  //lançar excepção
              }
          }
          else    //se o ano introduzido é negativo
          {
              throw new IllegalArgumentException();  // lançar excepção
          }
      }

      public  int DaysInTheMonth (int year, int month)  // Determinar o número de dias de cada mês
      {
          Boolean leapYear = (year % 4 == 0) && ( (year % 100 != 0) || (year % 400 == 0) );  // ano bissexto
          int maxDay=0;
          if (((month %2 !=0)&&(month < 8))||((month %2 ==0)&&(month > 7)))//meses com 31 dias
          {
              maxDay = 31;
              return maxDay;       // valor máximo de dias de cada mês
          }
          else if ((((month %2 ==0)&&(month!=2)&&(month < 8))||((month %2 !=0)&&(month > 7))))  // meses com 30 dias
          {
              maxDay = 30;
              return maxDay;       // valor máximo de dias de cada mês
          }
          else if (month==2) {  //mês (2) Fevereiro
              maxDay = (leapYear)?(29):(28);
              return maxDay;       // valor máximo de dias de cada mês
          }
          else
          {
              throw new IllegalArgumentException();  // lançar excepção
          }
      }
```

```
        /**
         * Constructor that allows Date to be called without any parameters.
         */
        public Date()
        {
           day = 0;
           month = 0;
           year = 0;
        }

        /**
         * Return the date as a String "DD-MM-YYYY"
         */
        public String toString(){ // Retorna a data com o seguinte formato "DD-MM-YYYY"
           String date = day + "-" + month + "-" + year;
           return date;
        }

        /**
         * Compares the claimed date with the date passed as parameter.
         * @param date
         * @return -1 , 0 or 1 if the claimed date is respectively less than , equal to or
greater than the date
         */

        public int compareTo(Date date){  //Compara a data invocada com a data passada por
parâmetro. Deve retornar <0, =0 ou >0, caso a data invocada seja respectivamente menor,
igual ou maior que a data "date".
           int yearDif = this.year - date.year;
           int monthDif = this.month - date.month;
           int dayDif = this.day - date.day;

           if ((yearDif==0)&&(monthDif==0)&&(dayDif==0)) // mesma data
           {
               return 0;
           }
           else if (((yearDif==0)&&(monthDif==0)&&(dayDif <
0))||((yearDif==0)&&(monthDif<0))||(yearDif<0))
           {
               return -1;  //Data invocada anterior à data "date"
           }
           else if (((yearDif==0)&&(monthDif==0)&&(dayDif >
0))||((yearDif==0)&&(monthDif>0))||(yearDif>0))
           {
               return 1;   //Data invocada anterior à data "date"
           }
        return 0;
        }

        /**
         * Add to date a certain number of days and returns a new date.
         * @param nDays
         * @return Date
         */
        public Date addDays(int nDays){ // Soma à data um determinado número de dias e
retorna uma nova data.
           int newDay = this.day;
```

```
            int newYear = this.year;
            int newMonth = this.month;
            int numberDays = nDays;

            if (numberDays < 0)
            {
                throw new IllegalArgumentException();  // lançar excepção
            }

            while (numberDays>=0)
            {
                if (numberDays == 0)    // A data que retorna é a mesma que já está em
sistema
                {
                   break;
                }
                else
                {
                   int monthDays = DaysInTheMonth (newYear, newMonth);
                   if (newDay + numberDays <= monthDays) // a nova data refere-se ao mesmo
mês e ano.
                   {
                       newDay = newDay + numberDays;
                       break;
                   }
                   else   // a nova data implica mudança do mês e/ou ano
                   {
                       numberDays = numberDays - (monthDays-newDay); // retirar dias até ao
final do corrente mês
                       newMonth ++;   //mudar de mês
                       newDay=1;
                       if (newMonth>12)  //mudança de ano
                       {
                          newMonth = 1;
                          newYear ++;
                       }
                   }
                }
            }
                return new Date (newYear,newMonth,newDay);
        }

/**
 * Return de difference in days between to dates
 * @param date
 * @return nDays
 */
       public int diffDays(Date date){  //Devolve a diferença em dias entre a data
invocada e outra data "date".
            int newDay = this.day;
            int newMonth = this.month;
            int newYear = this.year;
            int nDays = 0;

            switch (this.compareTo(date)){  //verificar qual é a data mais antiga
            case -1:              //a data introduzida é posterior à data invocada
                int monthDays1 = DaysInTheMonth (newYear, newMonth);
```

```
            if ((newMonth == date.month)&&(newYear == date.year)) // as duas datas
referem-se ao mesmo mês e ano.
                {
                    nDays = date.day - newDay;
                    break;
                }
                else   // a nova data implica mudança do mês e/ou ano
                {
                    nDays = monthDays1 - newDay ; // retirar dias até ao final do
corrente mês
                    newMonth ++;   //mudar de mês
                    newDay = 1;

                    while ((newMonth!=date.month)||(newYear!=date.year)) //faz ciclo
enquanto o mês e o ano da data invocada forem diferentes da data introduzida
                    {
                       nDays = nDays + monthDays1;
                       newMonth++;
                       if (newMonth>12)  //mudança de ano
                       {
                           newMonth = 1;
                           newYear ++;
                       }
                    }
                    nDays = nDays + date.day;
                    break;
                }
          case 0:
              nDays = 0;
              break;

          case 1://a data introduzida é anterior à data invocada
              int monthDays2 = DaysInTheMonth (newYear, newMonth);
              if ((newMonth == date.month)&&(newYear == date.year)) // as duas datas
referem-se ao mesmo mês e ano.
              {
                 nDays = newDay - date.day;
                 break;
              }
              else   // a nova data implica mudança do mês e/ou ano
              {
                 nDays = monthDays2 - date.day  ; // retirar dias até ao final do corrente
mês
                 date.month ++;   //mudar de mês

                 while ((newMonth!=date.month)||(newYear!=date.year))
                 {
                     nDays = nDays + monthDays2;
                     date.month ++;
                     if (date.month>12)  //mudança de ano
                     {
                            date.month = 1;
                            date.year ++;
                     }
                 }
                 nDays = nDays + newDay;
                 break;
              }
```

```
            }
            return nDays;
        }

        /**
         * @return day of the date
         */
        public int getDay(){
            return day;     //Retorna o dia da data.
        }

        /**
         * @return month of the date
         */
        public int getMonth() {
            return month;  //Retorna o mês da data.
        }

        /**
         * @return year of the date
         */
        public int getYear(){
            return year;    //Retorna o ano da data
        }
}
```

```
package Lab12;
import static org.junit.Assert.*;

import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;

public class BookTester {
        Book book;
        String publisher = "editora";
        String title = "título livro";

        @Before
        public void createBook(){ //antes de cada teste cria-se um objecto Book
            book = new Book (title, publisher);
        }

        /**
         * Unit Test for GetTitle
         */
        @Test
        public void GetTitleTest()
        {
            Assert.assertTrue(title.equals(book.getTitle()));
        }

        /**
         * Unit Test for GetPublisher
         */
        @Test
        public void GetPublisherTest()
        {
            Assert.assertTrue(publisher.equals(book.getPublisher()));
        }
```

```
        /**
         * Unit Test for Requisition's Date
         */
        @Test
        public void GetDateTest()
        {
           Date d1 = new Date(20,8,1999);
           book.requestIt(d1);

        Assert.assertTrue(d1.toString().equals(book.getLastRequestDate().toString()));
//transforma as duas datas em strings e compara-as
        }

        /**
         * Unit Test for RequestIt and ReturnIt Date
         */
        @Test
        public void GetReturnDateTest()
        {
           Assert.assertTrue(book.requestIt(new Date (5,3,2014)));
           Assert.assertTrue(!book.requestIt(new Date (5,8,2014)));
           Assert.assertTrue(book.returnIt());
        }

        /**
         * Unit test for number of requisition
         */
        @Test
        public void GetNumberRequestTest()
        {
           for (int i = 1; i <= 12; i++)
           {
               book.requestIt(new Date(20,i,2014));
               book.returnIt();
               int x = book.getTotalRequests();
               Assert.assertTrue(book.getTotalRequests() == i);
           }
        }

}
```

```
package Lab12;

import static org.junit.Assert.*;
import org.junit.Assert;
import org.junit.Before;
import org.junit.Test;


public class DateTester {
        Date d;
        @Before
        public void CreateDate()
        {
           d = new Date (20,10,2014);
        }

        /**
         * Unit test for an illegal argument in the parameter " year "
         */
        @Test(expected = IllegalArgumentException.class)    //indica ao compilador a
excepção esperada
```

```
    public void Dates_withNegativeYear_producesException()
    {
       new Date (10,5,-250);
    }

       /**
        * Unit test for an illegal argument in the parameter " month "
        */
       @Test(expected = IllegalArgumentException.class)   //indica ao compilador a
excepção esperada
    public void Dates_withMonthOutOfBounds_producesException()
    {
            new Date (10,15,2014);
    }

       /**
        * Unit test for an illegal argument in the parameter " day "
        */
       @Test(expected = IllegalArgumentException.class)   //indica ao compilador a
excepção esperada
    public void Dates_withDayOutOfBounds_producesException()
    {
            new Date (35,5,2014);
    }

       /**
        * Unit test to GetDay
        */
       @Test
       public void GetDayTest() {
            Assert.assertTrue(20== d.getDay());
       }

       /**
        * Unit test to GetMonth
        */
       @Test
       public void GetMonthTest() {
            Assert.assertTrue(10== d.getMonth());
       }

       /**
        * Unit test to GetYear
        */
       @Test
       public void GetYearTest() {
            Assert.assertTrue(2014== d.getYear());
       }

       /**
        * Unit test to ToString
        */
       public void ToStringTest ()
       {
            Assert.assertEquals("20-10-2014", d.toString());
       }

       /**
```

```
        * Unit test to compare two dates
        * Should return -1 , 0 or 1 if the claimed date is respectively less than , equal
to or greater than the date
        */
       public void CompareToTest()
       {
              Assert.assertTrue(-1 == d.compareTo(new Date(25,10,2014)));
              Assert.assertTrue(0 == d.compareTo(new Date(20,10,2014)));
              Assert.assertTrue(1 == d.compareTo(new Date(15,10,2014)));
       }

       /**
        * Unit test to add days to a date and return the new date
        * The examples test the change of the day, month and year
        */
       public void AddDaysTest()
       {
              Assert.assertTrue(new Date(25,10,2014)==d.addDays(5));
              Assert.assertTrue(new Date(9,11,2014)==d.addDays(20));
              Assert.assertTrue(new Date(3,1,2015)==d.addDays(75));
       }


       /**
        * Unit test for an illegal argument in the parameter " nDays " (negative)
        */
       @Test(expected = IllegalArgumentException.class)   //indica ao compilador a
excepção esperada
    public void Negative_NDays_producesException()
    {
              d.addDays(-5);
    }

       /**
        * Unit test to the difference of days between to dates
        */
       @Test
       public void diffDaysTest ()
       {
              Assert.assertTrue(5==d.diffDays(new Date(15,10,2014)));
              Assert.assertTrue(31==d.diffDays(new Date(20,9,2014)));
              Assert.assertTrue(20==d.diffDays(new Date(9,11,2014)));
              Assert.assertTrue(76==d.diffDays(new Date(3,1,2015)));
       }
}
```