Chapter 2

UML exercises

This chapter proposes small to medium scale exercises on UML. Some of them are by the author, whilst others have been taken from books (credits are made explicit in each exercise: where no explicit citation is given, the exercise is to be considered the author's work).

2.1 Use case diagrams

In this section we give some examples of use case diagrams for various situations.

2.1.1 Simplified ATM machine

Propose a use case diagram for an ATM machine for withdrawing cash. Make the use case simple yet informative; only include the major features.

2.1.1.1 Solution

The use case diagram is given in Fig. 2.1 (taken from [3], Fig. 16.5).

2.1.2 Vending machine

Propose a use case diagram for a vending machine that sells beverages and snacks. Make use of inclusion and extension associations, mark multiplicities and remember that a vending machine may need technical assistance from time to time.

2.1.2.1 Solution

The use case diagram is given in Fig. 2.2. We remark that the "+" character in front of multiplicities should not be there and that the {`xor`} constraint should be marked by a dashed segment rather than a box (the `umbrello` UML modeller that comes with the KDE linux desktop has some bugs and limitations).

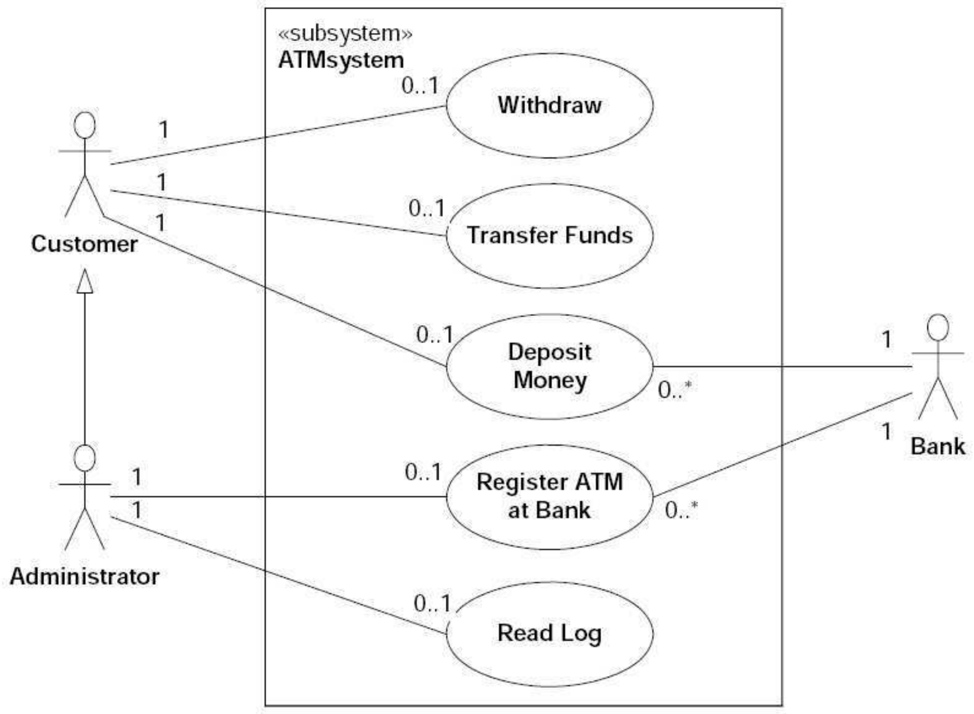

Figure 2.1: The simplified use case diagram of an ATM.

2.2 Sequence diagrams

In this section we shall present some easy examples of sequence diagrams.

2.2.1 The norm of a vector

Consider the following algorithm for computing the norm of a vector.

```
Class Array {
  ...
  public:

    // return the index-th component of the array
    double get(int index);
  ...
};

double norm(const Array& myArray) {
  double theNorm = 0;
  for(int index = 0; index < myArray.size() - 1; index++) {
    theNorm = theNorm + myArray.get(index);
  }
  theNorm = sqrt(theNorm);
  return theNorm;
```

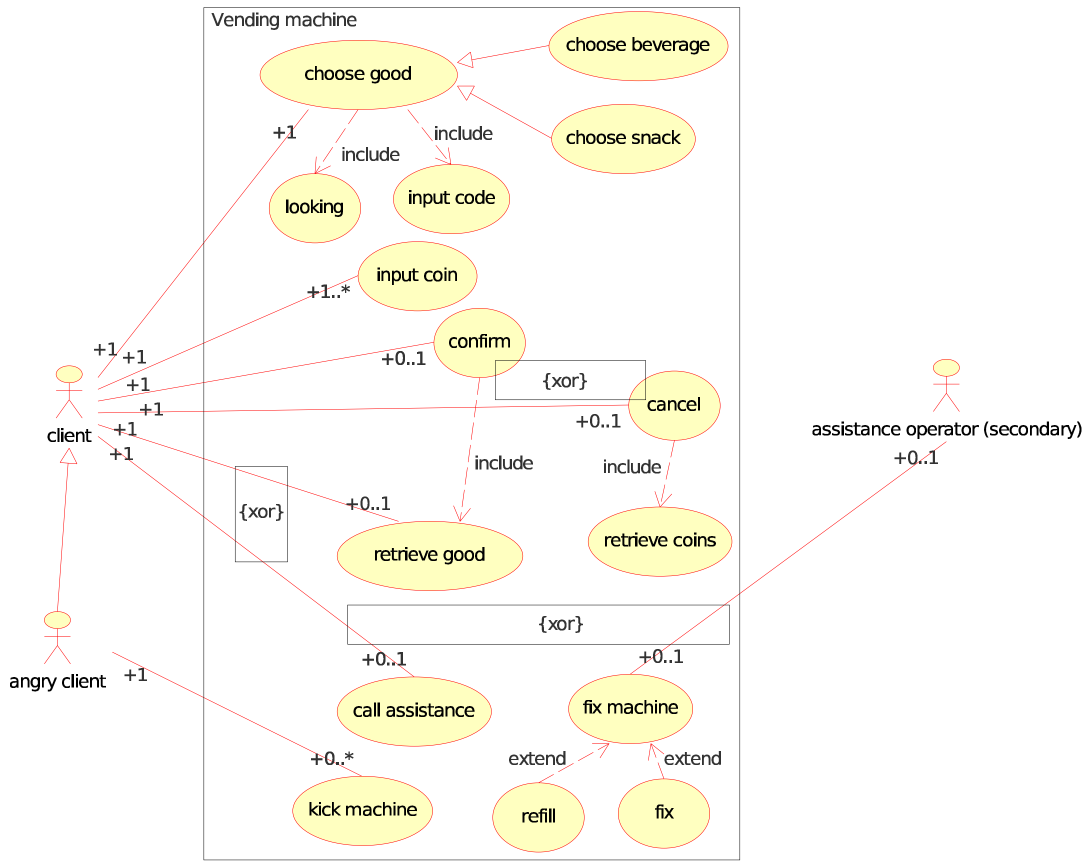

Figure 2.2: The use case diagram of a vending machine.

```
}
```

Write down a sequence diagram that describes the `norm()` function.

2.2.1.1 Solution

The sequence diagram in Fig. 2.3 describes the `norm()` function.

2.2.2 Displaying graphical objects

Write a sequence diagram for a program that displays Fig. 2.4 on the screen in the order left → right.

2.2.2.1 Solution

The sequence diagram in Fig. 2.5 describes the required behaviour.

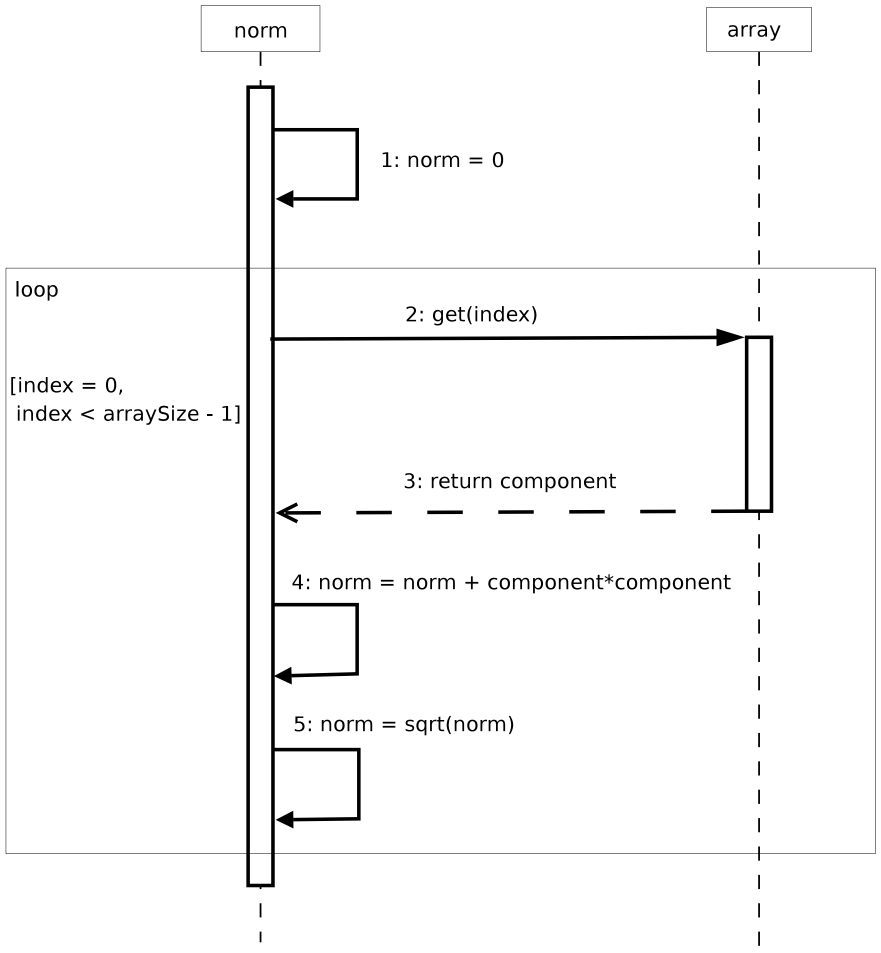

Figure 2.3: The sequence diagram describing the computation of the norm of a vector.

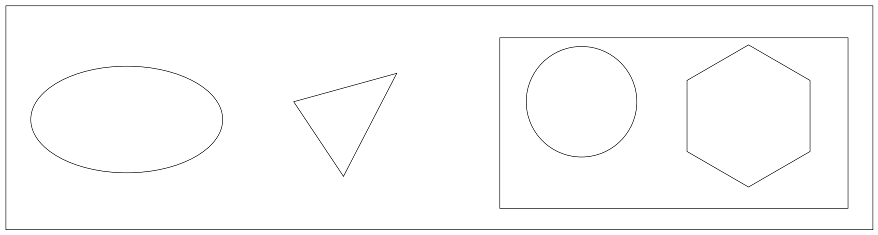

Figure 2.4: The sequence diagram describing the computation of the norm of a vector.

2.2.3 Vending machine

Draw a sequence diagram for the vending machine of Sect. 2.1.2.

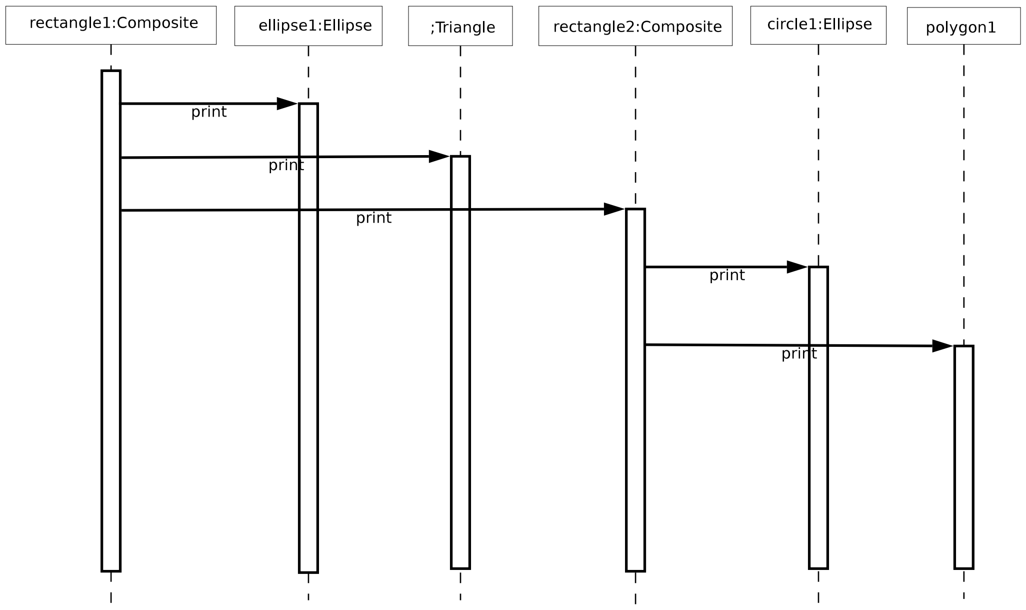

Figure 2.5: The sequence diagram describing the printing of the drawing in Fig. 2.4.

2.2.3.1 Solution

The sequence diagram in Fig. 2.6 describes the required behaviour.

2.3 Class diagrams

In this section we present some elementary exercises on class diagrams.

2.3.1 Complex number class

Draw a class diagram for the single class `Complex`. A `Complex` object has a private real and an imaginary part (of type `double`), and can perform addition, subtraction, multiplication and division by another complex number.

2.3.1.1 Solution

The class diagram for the `Complex` class is given in Fig. 2.7.

Most UML modellers can be used to automatically generate C++ (or Java) code from the class diagram. This results in "class skeleton" files (a header file `Complex.h` and a corresponding implementation file `Complex.cpp`). Both are given below.

```
/************************************************************************
                        Complex.h.h - Copyright liberti

Here you can write a license for your code, some comments or any other
information you want to have in your generated code. To to this simply
```

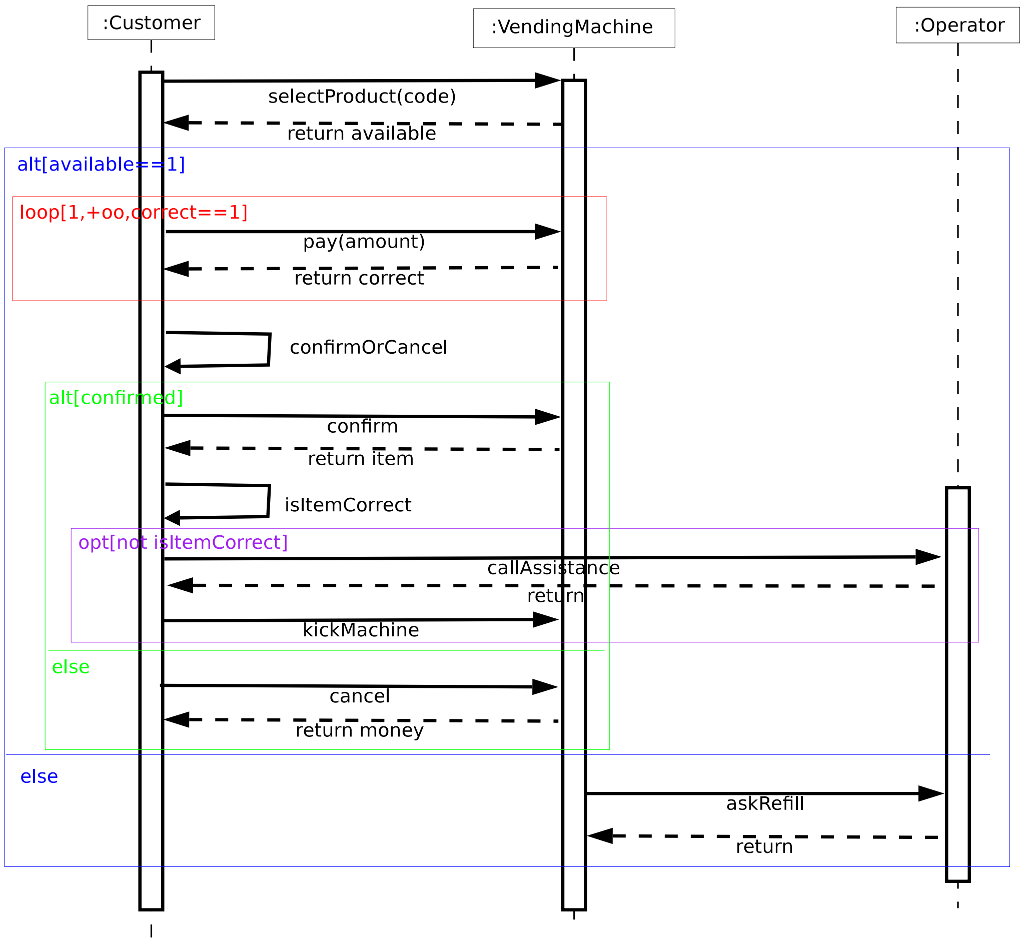

Figure 2.6: The sequence diagram for the vending machine of Sect. 2.1.2.

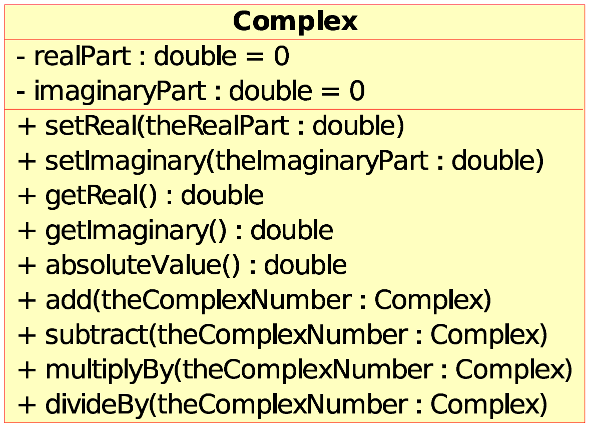
Complex
- realPart : double = 0
- imaginaryPart : double = 0
+ setReal(theRealPart : double)
+ setImaginary(theImaginaryPart : double)
+ getReal() : double
+ getImaginary() : double
+ absoluteValue() : double
+ add(theComplexNumber : Complex)
+ subtract(theComplexNumber : Complex)
+ multiplyBy(theComplexNumber : Complex)
+ divideBy(theComplexNumber : Complex)

Figure 2.7: The class diagram for a complex number.

```
configure the "headings" directory in uml to point to a directory
where you have your heading files.
```

```
or you can just replace the contents of this file with your own.
If you want to do this, this file is located at

/usr/share/apps/umbrello/headings/heading.h

-->Code Generators searches for heading files based on the file extension
   i.e. it will look for a file name ending in ".h" to include in C++ header
   files, and for a file name ending in ".java" to include in all generated
   java code.
   If you name the file "heading.<extension>", Code Generator will always
   choose this file even if there are other files with the same extension in the
   directory. If you name the file something else, it must be the only one with that
   extension in the directory to guarantee that Code Generator will choose it.

you can use variables in your heading files which are replaced at generation
time. possible variables are : author, date, time, filename and filepath.
just write %variable_name%

This file was generated on %date% at %time%
**************************************************************************/


#ifndef COMPLEX_H
#define COMPLEX_H

#include <string>

/**
  * class Complex
  */

class Complex
{
public:

  // Constructors/Destructors
  //


  /**
   * Empty Constructor
   */
  Complex ( );

  /**
   * Empty Destructor
   */
  virtual ~Complex ( );

  // Static public attributes
  //

  // public attributes
  //


  // public attribute accessor methods
  //


  // public attribute accessor methods
  //



  /**
   * @param  theRealPart
   */
  void setReal (double theRealPart );


  /**
   * @param  theImaginaryPart
   */
  void setImaginary (double theImaginaryPart );


  /**
   * @return double
   */
  double getReal ( );


  /**
   * @return double
   */
  double getImaginary ( );


  /**
   * @return double
   */
  double absoluteValue ( );


  /**
   * @param  theComplexNumber
   */
  void add (Complex theComplexNumber );


  /**
   * @param  theComplexNumber
   */
  void subtract (Complex theComplexNumber );


  /**
   * @param  theComplexNumber
   */
  void multiplyBy (Complex theComplexNumber );


  /**
   * @param  theComplexNumber
   */
  void divideBy (Complex theComplexNumber );

protected:

  // Static protected attributes
  //

  // protected attributes
  //

public:


  // protected attribute accessor methods
  //

protected:

public:


  // protected attribute accessor methods
  //

protected:


private:

  // Static private attributes
  //

  // private attributes
  //

  double m_realPart;
  double m_imaginaryPart;
public:


  // private attribute accessor methods
  //

private:
```

```
public:


  // private attribute accessor methods
  //


  /**
   * Set the value of m_realPart
   * @param new_var the new value of m_realPart
   */
  void setRealPart ( double new_var );

  /**
   * Get the value of m_realPart
   * @return the value of m_realPart
   */
  double getRealPart ( );


  /**
   * Set the value of m_imaginaryPart
   * @param new_var the new value of m_imaginaryPart
   */
  void setImaginaryPart ( double new_var );

  /**
   * Get the value of m_imaginaryPart
   * @return the value of m_imaginaryPart
   */
  double getImaginaryPart ( );

private:


  void initAttributes ( ) ;

};

#endif // COMPLEX_H



/************************************************************************
                        Complex.h.cpp - Copyright liberti

Here you can write a license for your code, some comments or any other
information you want to have in your generated code. To to this simply
configure the "headings" directory in uml to point to a directory
where you have your heading files.

or you can just replace the contents of this file with your own.
If you want to do this, this file is located at

/usr/share/apps/umbrello/headings/heading.cpp

-->Code Generators searches for heading files based on the file extension
   i.e. it will look for a file name ending in ".h" to include in C++ header
   files, and for a file name ending in ".java" to include in all generated
   java code.
   If you name the file "heading.<extension>", Code Generator will always
   choose this file even if there are other files with the same extension in the
   directory. If you name the file something else, it must be the only one with that
   extension in the directory to guarantee that Code Generator will choose it.

you can use variables in your heading files which are replaced at generation
time. possible variables are : author, date, time, filename and filepath.
just write %variable_name%

This file was generated on %date% at %time%
**************************************************************************/

#include "Complex.h"

// Constructors/Destructors
//

Complex::Complex ( ) {
initAttributes();
}

Complex::~Complex ( ) { }
```

```
//
// Methods
//


// Accessor methods
//


// public static attribute accessor methods
//


// public attribute accessor methods
//


// protected static attribute accessor methods
//


// protected attribute accessor methods
//


// private static attribute accessor methods
//


// private attribute accessor methods
//


/**
 * Set the value of m_realPart
 * @param new_var the new value of m_realPart
 */
void Complex::setRealPart ( double new_var ) {
  m_realPart = new_var;
}

/**
 * Get the value of m_realPart
 * @return the value of m_realPart
 */
double Complex::getRealPart ( ) {
  return m_realPart;
}

/**
 * Set the value of m_imaginaryPart
 * @param new_var the new value of m_imaginaryPart
 */
void Complex::setImaginaryPart ( double new_var ) {
  m_imaginaryPart = new_var;
}

/**
 * Get the value of m_imaginaryPart
 * @return the value of m_imaginaryPart
 */
double Complex::getImaginaryPart ( ) {
  return m_imaginaryPart;
}

// Other methods
//


/**
 * @param  theRealPart
 */
void Complex::setReal (double theRealPart ) {

}


/**
 * @param  theImaginaryPart
 */
```

```
void Complex::setImaginary (double theImaginaryPart ) {

}


/**
 * @return double
 */
double Complex::getReal ( ) {

}


/**
 * @return double
 */
double Complex::getImaginary ( ) {

}


/**
 * @return double
 */
double Complex::absoluteValue ( ) {

}


/**
 * @param  theComplexNumber
 */
void Complex::add (Complex theComplexNumber ) {

}


/**
 * @param  theComplexNumber
 */
void Complex::subtract (Complex theComplexNumber ) {

}


/**
 * @param  theComplexNumber
 */
void Complex::multiplyBy (Complex theComplexNumber ) {

}


/**
 * @param  theComplexNumber
 */
void Complex::divideBy (Complex theComplexNumber ) {

}

void Complex::initAttributes ( ) {
  m_realPart = 0;
  m_imaginaryPart = 0;
}
```

2.3.2 Singly linked list

Draw a class diagram representing a singly linked list.