Fundamental Concepts of Programming Languages

Object-Oriented Programming Languages

Lecture 10

conf. dr. ing. Ciprian-Bogdan Chirila

University Politehnica Timisoara
Department of Computing and Information Technology

November 22, 2022

FCPL - 10 - Object-Oriented Programming Languages

1. Introduction to OOP
2. Object oriented programming
   - Inheritance
   - Dynamic binding
3. Object-oriented programming in Java
   - General aspects
   - Examples
4. Object-oriented programming in C#
5. Bibliography

Introduction to OOP

- Objects
- Object-orientation
- Attached to each software development phase
  - Design
  - Development
  - Testing
- Software environment components
  - Databases
  - Operating systems
  - IDEs

Introduction to OOP

- Software development starts from real world objects identification
- Solving the problem means creating a model to that reality
- The model will contain object interacting between them
- The objects are models of real world objects and their interacting operations

Introduction to OOP

- Object description is made through abstract data types
- Object based programming - no inheritance
- The program is organized on a set of objects described by abstract data types
- Object based programming languages
  - Ada, Simula 67, Modula 2

FCPL - 10 - Object-Oriented Programming Languages

1. Introduction to OOP
2. Object oriented programming
   - Inheritance
   - Dynamic binding
3. Object-oriented programming in Java
   - General aspects
   - Examples
4. Object-oriented programming in C#
5. Bibliography

Object oriented programming

- The central concept is the object
- The object has
  - Own state – local
  - Behavior – set of methods
- Applying methods the object state changes
- Object state is defined through its variables
- Implicitly inaccessible by outside
- Through external accessible methods the objects interact one with the other

The object

- Is an instantiation of a class
- The class is a type constructor describing variables and methods of objects instantiated through that class
- Class programming is the fundamental concept in object based programming
- Object-oriented programming has two extra features:
  - Inheritance
  - Dynamic binding

Inheritance

- The feature allowing to describe new classes which take:
  - the state variables
  - the behavior functions
- The new class is called subclass of the original one
- The old class is called superclass for the new defined one

Inheritance

- Allows to define a class without writing it from completely from scratch
- This is valid when there is already another class with common characteristics
- It is possible for the superclass to have:
  - New attributes
  - New methods
  - Redefined attributes
  - Redefined methods
- The concept is called specialization

Example in C++

- A stack which implements the push operation for a pair of elements
- The new class will be implemented as a an extension of the class `Stack`

Example class Stack

```
#ifndef __STACK_H
#define __STACK_H
class Stack
{
private:
    int nr_max;
protected:
int *tab_st;
int ind;
public:
  Stack(int);
  void push(int x);
  int pop();
  int top();
  int empty();
  int overflow();
};
#endif
```

Example class Stack

```
#include "Stack.h"
#include <iostream>
using namespace std;
Stack::Stack(int nr_max)
{
    this->nr_max=nr_max;
    tab_st=new int[nr_max];
ind=0;
}
int Stack::pop()
{
if(!empty())
  {
   ind--;
   return tab_st[ind];
  }
  return -1;
}
```

Example class Stack

```
void Stack::push(int x)
{
 if(!overflow())
 {
  tab_st[ind]=x;
  ind++;
 }
}
int Stack::top()
{
 if(!empty())
 {
  return tab_st[ind-1];
 }
 return -1;
}
```

Example class Stack

```
int Stack::empty()
{
 return ind==0;
}

int Stack::overflow()
{
 return ind >= nr_max;
}
```

Example using references

```
int main()
{
 Stack st1(100);Stack st2(100);Stack st3(55);
 st1.push(10);
 //--------------
 st2.push(20);
 //-------------
 int i=st1.top();
 cout<<i<<endl;
 //------------
 int j=st2.pop();
 cout<<j<<endl;
}
```

Example class StackSpec

```
#ifndef __STACKSPEC_H
#define __STACKSPEC_H
class StackSpec : public Stack
{
public:
 StackSpec(int nr_max);
 void push_2(int x,int y);
};
#endif
```

Example class StackSpec

```
#include "Stack.h"
#include "StackSpec.h"
#include <iostream>
using namespace std;
StackSpec::StackSpec(int nr_max):Stack(nr_max)
{
}
void StackSpec::push_2(int x,int y)
{
 push(x);
 push(y);
}
```

Example class StackSpec

```
int main()
{
 StackSpec st(150);
 st.push(15);
 st.push_2(155,25);

 int i=st.pop();
 cout<<i<<endl;

 int j=st.pop();
 cout<<j<<endl;

 int k=st.pop();
 cout<<k<<endl;

 return 0;
}
```

Example class StackSpecSpec

```
#ifndef __STACKSPECSPEC_H
#define __STACKSPECSPEC_H
#include "StackSpec.h"
class StackSpecSpec : public StackSpec
{
public:
 StackSpecSpec(int);
 int under_top();
 int pop();
 int top();
 int empty_spec();
};
#endif
```

Example class StackSpecSpec

```
#include "Stack.h"
#include "StackSpec.h"
#include "StackSpecSpec.h"
#include <iostream>
using namespace std;
StackSpecSpec::StackSpecSpec(int nr_max):StackSpec(nr_max)
{
}
int StackSpecSpec::pop()
{
 if(!empty())
 {
  ind--;
  return tab_st[ind];
 }
 return -1;
}
```

Example class StackSpecSpec

```
int StackSpecSpec::top()
{
 if(!empty())
  return tab_st[ind-1];
 else
  return -1;
}
int StackSpecSpec::under_top()
{
 if(!empty_spec())
  return tab_st[ind-2];
 return -1;
}
int StackSpecSpec::empty_spec()
{
 return ind<2;
}
```

Example class StackSpecSpec

```
int main()
{
 Stack st1(100);
 StackSpec st2(50);
 StackSpecSpec st3(80);
 int i;
 //i=st1.under_top(); --ilegal
 //i=st2.under_top(); --ilegal
 i=st3.under_top();
 cout<<i<<endl;

 //st1.push_2(150,12); --ilegal
 st2.push_2(11,110);
 st3.push_2(5,17);
```

Example class StackSpecSpec

```
 i=st1.top(); //if the stack is empty, -1 is returned
 cout<<i<<endl;
 i=st2.top(); //if the stack is empty, -1 is returned
 cout<<i<<endl;
 i=st3.top(); //if the stack is empty, -1 is returned
 cout<<i<<endl;
}
```

Dynamic binding

```
int main()
{
 Stack *pst=NULL;
 pst=new Stack(100);
 i=pst->top();
 cout<<i<<endl;

 pst=new StackSpecSpec(100);
 i=pst->top();
 cout<<i<<endl;
}
```

FCPL - 10 - Object-Oriented Programming Languages

1. Introduction to OOP
2. Object oriented programming
   - Inheritance
   - Dynamic binding
3. Object-oriented programming in Java
   - General aspects
   - Examples
4. Object-oriented programming in C#
5. Bibliography

Object-oriented programming in Java

- The project was launched at Sun in 1990
- A PL for domestic electronic devices
- The goal was to make programs live on different hardware architectures
- Starting from 1993 with www development Java was designed such that applications to be executed on any computer connected to the Internet independently of its architecture

The Java PL

- Took a lot of syntax from C and C++
- Some features were eliminated because of security reasons
- No pointers allowed
- No explicit memory deallocations
- No more multiple inheritance
- Portability is based on
  - Java compiler generating byte code
  - The byte code can be executed on any machine having a virtual machine on the top of it

Comparisons with other PLs

- Inspired more from SmallTalk and Eiffel Less from C++
- SmallTalk is an extreme OOPL
- It operates only on objects even for basic types
- An integer is an object
- The operations between objects are sent as messages

Comparisons with other PLs

- C++ is a usual data oriented PL retrofitted with object-oriented features
- In Java
  - all entities are objects
  - Except primitive types (integer, real, character, boolean)
- Data can be accessed directly by name
- Objects
  - can be accessed indirectly references
  - must be created explicitly by new operations

Java libraries

- Rich set of predefined classes
- Rapid application development
- Windowing toolkit
  - Windows
  - Dialogs
  - Animated graphics
  - Network remote connections
  - Mouse events listeners
- Inheritance and redefinition allows adaptation of classes to application specific needs

Java example

- Java program is a set of classes
- One class must contain a main method
- Needed as program starting point

Java example

```
public class Circle
{
 protected double x, y, r;
 public static int num_circles = 0;
 public Circle(double x, double y, double r)
 {
  this.x=x; this.y=y; this.r=r;
  num_circles++;
 }
```

Java example

```
public Circle(double r)
{
 this(0.0, 0.0, r);
}
public Circle(Circle c)
{
 this(c.x, c.y, c.r);
}
public Circle()
{
 this(0.0, 0.0, 1.0);
}
```

Java example

```
public double circumference()
{
 return 2*3.14159*r;
}

public double area()
{
 return 3.14159*r*r;
}
```

Java example

```
import java.awt.*;
public class GraphicCircle extends Circle
{
 protected Color outline, fill;
 public GraphicCircle(double x, double y, double r,  Color outline,
 {
  super(x, y, r);
  this.outline=outline;
  this.fill=fill;
 }
```

Java example

```
 public GraphicCircle(Color outline, Color fill)
 {
  this.outline=outline;
  this.fill=fill;
 }

 public void draw(DrawWindow dw)
 {
  dw.drawCircle(x,y,r,outline,fill);
 }
}
```

Java example

```
import java.awt.*;
public class GraphicCircleSmart extends GraphicCircle
{
 public GraphicCircleSmart(double x, double y,
 double r,Color outline, Color fill)
 {
  super(x, y, r,outline,fill);
 }

 public GraphicCircleSmart(Color outline, Color fill)
 {
  super(outline,fill);
 }
```

Java example

```
public void draw(DrawWindow dw)
{
 super.draw(dw);
 dw.drawLine(x-r, y, x+r, y, outline);
 dw.drawLine(x, y+r, x, y-r, outline);
}
```

Java example

```
 public put_outline(Color outline)
 {
  this.outline=outline;
 }
 public put_fill(Color fill)
 {
  this.fill=fill;
 }
}
```

Java example

```
public class DrawWindow
{
 public drawCircle(double x, double y, double r,Color outline,
  Color fill)
 {
 }

 public drawLine(double x1, double y1, double x2,double y2,
  Color outline)
 {
 }
}
```

Java example

```
import java.awt.*
public class TheMain
{
 public static void main (String args[])
 {
  Color forOutline=new Color( -------);
  Color forFill=new Color( -------);
  DrawWindow theFirstWd=new DrawWindow(-------);
  DrawWindow theSecondWd=new DrawWindow(-------);
```

Java example

```
Color cl;
double total_area=0;
GraphicCircle tabCircle[]=new GraphicCircle[4];
DrawWindow tabWindow[]=new DrawWindow[4];
tabCircle[0]=new GraphicCircle(1.0, 1.0, 1.0, forOutline, forFill);
tabWindow[0]=theFirstWd;
```

Java example

```
tabCircle[1]=new GraphicCircleSmart(forOutline,forFill);
tabWindow[1]=theSecondWd;

tabCircle[2]=new GraphicCircle(forOutline,forFill);
tabWindow[2]=theFirstWd;

tabCircle[3]=
new GraphicCircleSmart(5.0, 7.0, 3.0, forOutline,forFill);
tabWindow[3]=theSecondtWd;
```

Java example

```
for(int i=0; i<tabCircle.length; i++)
{
 total_area+=tabCircle[i].area();
 tabCircle[i].draw(tabWindow[i]);
}

cl=new Color (--------);
tabCircle[1].put_outline(cl);  //illegal
((GraphicCircleSmart)tabCircle[1]).put_online(cl);
tabCircle[1].draw(theFirstWd);
```

Comments on the Java example

- The example is a Java application
- Class Circle implements
  - circle attributes
  - methods
    - Circumference
    - Area

Constructors

- 3 constructors
  - Methods having the same name as the class
  - Executed when an object is created
  - Activation is made according to constructor signature
  - If none defined there is an implicit one
    - with no parameters
    - empty body
    - just memory allocation

Variables and methods

- modifiers apply to class members
  - Variables and methods
- private modifier
  - Visible only in the class
- public modifier
  - Visible from everywhere

Variables and methods

- non-static attributes (dynamic)
  - object associated
  - methods are the same for all objects, but data is not
- static members
  - class associated
  - not object associated !!!
  - num_circles count the number of class instantiations

Inheritance

- GraphicCircle subclass of Circle
- extends the draw() method
- Draws the circle in a window given as parameter
- The window is modelled by DrawWindow class
- colors are modelled by java.awt.Color
- awt – Abstract Windowing Toolkit
- Swing, SWT - Standard Widget Toolkit
- JavaFX for Rich Internet Applications

Inheritance

- GraphicCircleSmart subclass of GraphicCircle
- Extends the class with put_outline and put_fill
- Redefined the drawing method
- Class "TheMain" holds method "main"
- Two arrays present treated as objects
- Array size accessed by a field called "length"
- At array creation each element is null
- The array holds only references to objects

FCPL - 10 - Object-Oriented Programming Languages

1. Introduction to OOP
2. Object oriented programming
   - Inheritance
   - Dynamic binding
3. Object-oriented programming in Java
   - General aspects
   - Examples
4. Object-oriented programming in C#
5. Bibliography

Object-oriented programming in C#

- Built by Microsoft for .NET platform
- Like Java is derived from C and C++
- Portability concept taken from Java
  - Based on Microsoft intermediate language – MSIL
  - .NET framework as virtual machine
- Programming in multiple PLs
- Each part of the program can be written in the most expressive PL
- Full integration with the Windows platform

Object-oriented programming in C#

- C# programs start in Main function
- C# system contain classes
- Organized in hierarchies by inheritance
- Regarding inheritance
  - Superclass or base class
  - Subclass or derived class
- Multiple inheritance is not allowed
  - A class may not have multiple superclasses

Example in C#

```
using System;

// A class representing bi-dimensional objects.
class Shape2D
{
 public double width;
 public double height;
 public void showDim()
 {
  Console.WriteLine("Width and height are " +
  width +" and " + height);
 }
}
```

Example in C#

```
// class Triangle derives from class Shape2D.
class Triangle : Shape2D
{
 public string typeOfTriangle;
 public double area()
 {
  return width * height / 2;
 }

 public void showType ()
 {
  Console.WriteLine("Triangle is " + typeOfTriangle);
 }
}
```

Example in C#

```
class Shapes
{
 public static void Main()
 {
  Triangle t1 = new Triangle();
  t1.width = 4.0;
  t1.height = 4.0;
  t1.typeOfTriangle = "isosceles";

  Triangle t2 = new Triangle();
  t2.width = 8.0;
  t2.height = 12.0;
  t2.typeOfTriangle = "rectangular";
```

Example in C#

```
  ConsoleWriteln("Information about t1: ");
  t1.showType();
  t1.showDim() ;

  ConsoleWriteLine("Triangle area is " + t1.aria());
  ConsoleWriteLine();

  ConsoleWriteln("Information about t2: ");
  t2.showType();
  t2.showDim() ;
  ConsoleWriteLine("Triangle area is " + t2.aria());
 }
}
```

Object-oriented programming in C#

- Class Shape2D
  - Defines the generic shape attributes
  - Square
  - Rectangle
  - Triangle
- Class Triangle
  - Is derived from Shape2D
  - One attribute added: typeOfTriangle
  - two methods added: area(), showType()
  - Is able to refer attributes of Shape2D as its own

FCPL - 10 - Object-Oriented Programming Languages

1. Introduction to OOP
2. Object oriented programming
   - Inheritance
   - Dynamic binding
3. Object-oriented programming in Java
   - General aspects
   - Examples
4. Object-oriented programming in C#
5. Bibliography

Bibliography

1. Horia Ciocarlie – The programming language universe, second edition, Timisoara, 2013.
2. Carlo Ghezzi, Mehdi Jarayeri – Programming Languages, John Wiley, 1987.
3. Ellis Horrowitz – Fundamentals of programming languages, Computer Science Press, 1984.
4. Donald Knuth – The art of computer programming, 2002.