Fundamentals of Programming Languages

Object Oriented Programming Languages

Lecture 10

sl. dr. ing. Ciprian-Bogdan Chirila

Lecture outline

- Object-oriented programming
  - Inheritance
  - Dynamic binding
- Object-oriented programming in Java
  - General aspects
  - Example
- Object-oriented programming in C#
- Object-oriented programming in Lisp
  - Defining object and accessing their components
  - Slot inheritance
  - Method inheritance and message passing

Introduction to OOP

- Objects
- Object-oriented
- 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
- The program is organized on a set of objects described by abstract data types
- Object based programming languages
  - Ada, Simula 67, Modula 2

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

- 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** defined in the previous lecture

Example

```
type stack_spec(nr_max:integer) extends stack=class
        operations push_2;
        procedure push_2(x,y:integer);
        begin
                push(x);
                push(y);
        end;

begin
end;
```

Example – using references

```
var
        st : stack_spec;
-----------------------------------
        st := new stack_spec(150);
-----------------------------------
        st.push(15);
        st.push_2(155,25);
```

Example – under_top()

```
type stack_spec_spec(nr_max:integer) extends
stack_spec=class
        operations under_top,top,pop;

        function top():integer;
        begin
                if not empty() then
                        return tab_st[ind-1];
                else
                        return –1;
                end if;
        end;
```

Example – under_top()

```
function pop():integer;
        begin
                if not empty() then
                        ind:=ind-1;
                        return tab_st[ind];
                else
                        return –1;
                end if;
        end;
```

Example – under_top()

```
function under_top():integer;
        begin
                if not empty_spec() then
                        return tab_st[ind-2];
                else return –1;
                end if;
        end;
```

Example – under_top()

```
function empty_spec():boolean;
        begin
                return ind <= 2;
        end;
```

Example – under_top()

```
var
st1 : stack = new stack(100);
st2 : stack_spec = new stack_spec(50);
st3 : stack_spec_spec =
        new stack_spec_spec(80);
i: integer;
i := st1.under_top(); --ilegal
i := st2.under_top(); --ilegal
i := st3.under_top();
```

Example – under_top()

st1.push_2(150,12); --ilegal

st2.push_2(11,110);

st3.push_2(5,17);

i := st1.top(); --if the stack is empty, an exception is generated

i := st2.top(); --if the stack is empty, an exception is generated

i := st3.top(); --if the stack is empty, -1 is returned

Dynamic binding

```
st : stack;
i : integer;
with_exception : boolean;
----------------------
if with_exception then
        st:=new stack(100);
else
        st:=new stack_spec_spec(100);
end if;
----------------------
i:=st.top();
```

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 PL data oriented 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
- For Applets there is no main
- An instance of java.applet.Applet is needed
- The applet
  - subclass must contain init() and paint() methods
  - is included in a web page
  - runs in a browser

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);
}
```

Java Example

```
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, Color fill)
        {
                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 the_main
{
        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(init I=0; I<tabCircle.length; I++)
{
        total_area+=tabCircle[i].area();
        tabCircle[i].draw(tabWindow[i]);
}
```

Java example

```
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

Inheritance

- GraphicCircleSmart subclass of GraphicCircle
- Extends the class with put_outline and put_fill
- Redefined the drawing method
- Class “the_main” 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

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

```
// 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

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

Example

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

Example

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

Example

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

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

Comments

- 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