Object Oriented Programming

Lecture No. 4

Ing. Lukáš Slánský

Univerzita Pardubice

3. 11. 2008

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Contents

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Relations between objects

"Object can contain other objects".

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.
  - Dog is meaningless without head.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.
  - Dog is meaningless without head. And head is meaningless without dog.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.
  - Dog is meaningless without head. And head is meaningless without dog.
- Object can have other object as the essence of its' existence –

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.
  - Dog is meaningless without head. And head is meaningless without dog.
- Object can have other object as the essence of its' existence – *inheritance*.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.
  - Dog is meaningless without head. And head is meaningless without dog.
- Object can have other object as the essence of its' existence – *inheritance*.
  - Interaction just like between

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.
  - Dog is meaningless without head. And head is meaningless without dog.
- Object can have other object as the essence of its' existence – *inheritance*.
  - Interaction just like between vehicle-car.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.
  - Dog is meaningless without head. And head is meaningless without dog.
- Object can have other object as the essence of its' existence – *inheritance*.
  - Interaction just like between vehicle-car.
  - Inner object (vehicle) can exist without outer one (car).

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Relations between objects

"Object can contain other objects".

- Object uses other object – *association*.
  - Interaction just like between car-driver.
  - Both object can exist (and have sense) even without the other.
- Object contain other objects – *aggregation*.
  - I. e. crowd-man. The crowd contains (is aggregation) unknown number of mans.
  - Both object can exist without the other. Outer object (crowd) can loose some functionality without the inner (man).
  - The inner object can be part of several aggregates.
- Object is composed of other objects – *composition*.
  - Interaction just like between – dog-head.
  - Dog is meaningless without head. And head is meaningless without dog.
- Object can have other object as the essence of its' existence – *inheritance*.
  - Interaction just like between vehicle-car.
  - Inner object (vehicle) can exist without outer one (car).Outer object is meaningless without inner one.

How to Distinguish Objects?

1. Describe the functionality of program (or module, component, . . .) in common language.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

How to Distinguish Objects?

1. Describe the functionality of program (or module, component, . . .) in common language.
2. Every noun is (or should be) an Object.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

How to Distinguish Objects?

1. Describe the functionality of program (or module, component, . . .) in common language.
2. Every noun is (or should be) an Object.
3. Every verb describes method and/or relation.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

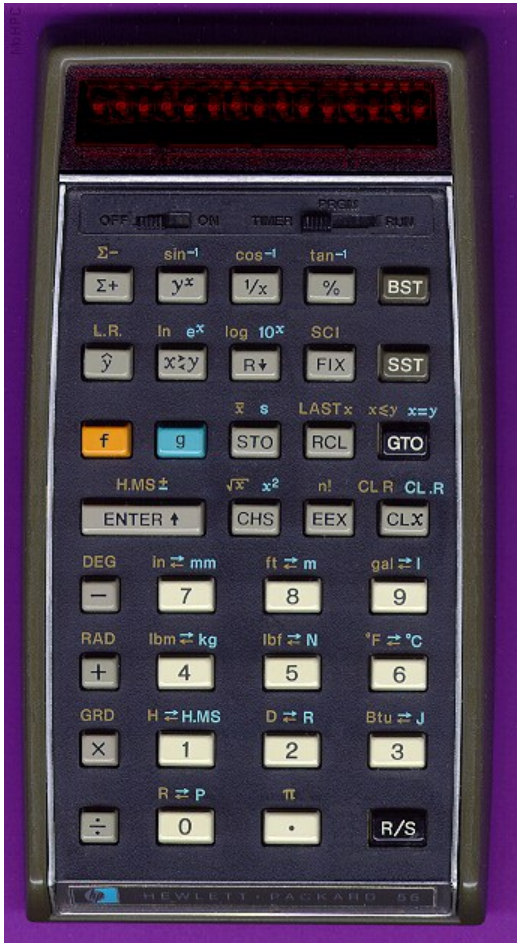
Example of Objects Composition

Design calculator that will work according to reverse polish notation. It can process basic math functions (+, -, *, /).

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition

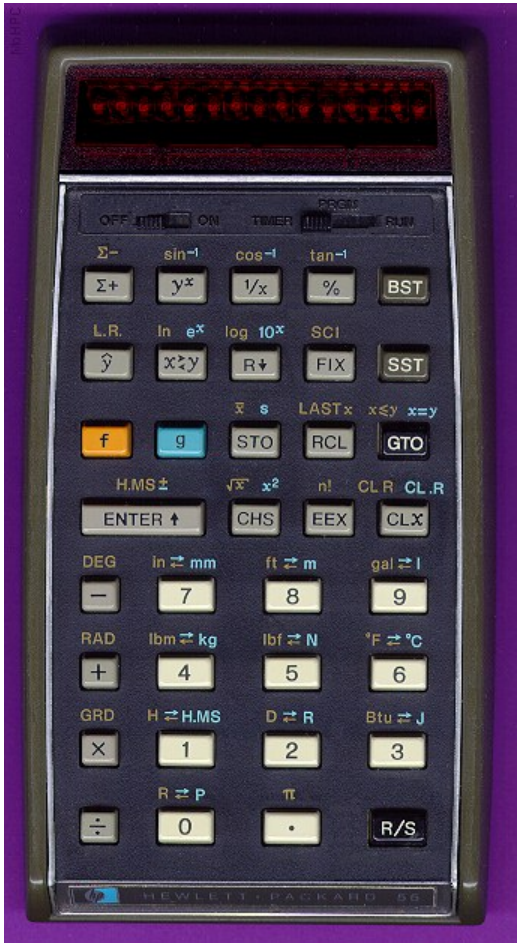
Design calculator that will work according to reverse polish notation. It can process basic math functions (+, -, *, /).



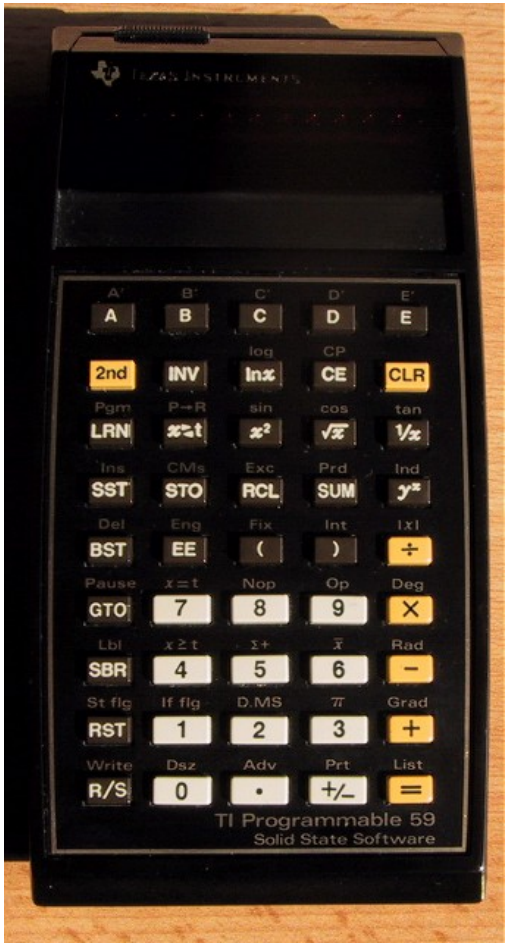
HP-55

Example of Objects Composition

Design calculator that will work according to reverse polish notation. It can process basic math functions (+, -, *, /).



HP-55

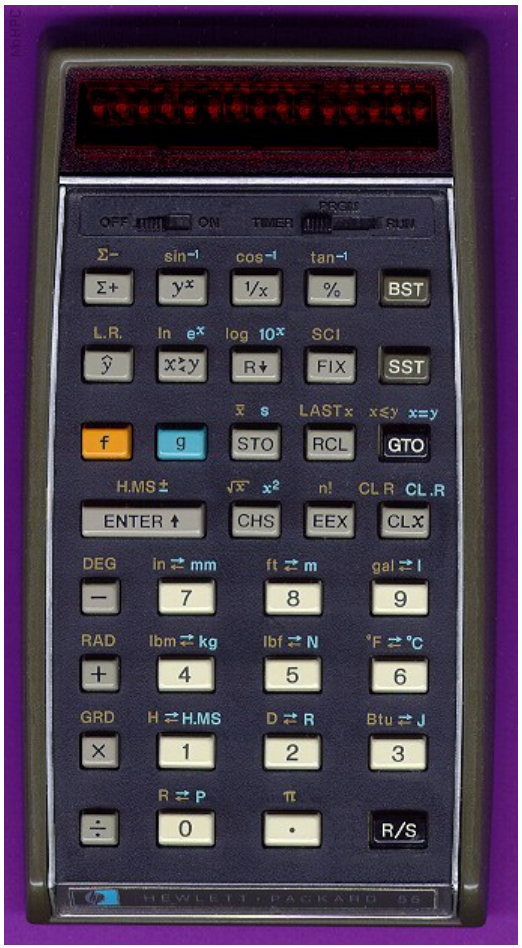


TI-59

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

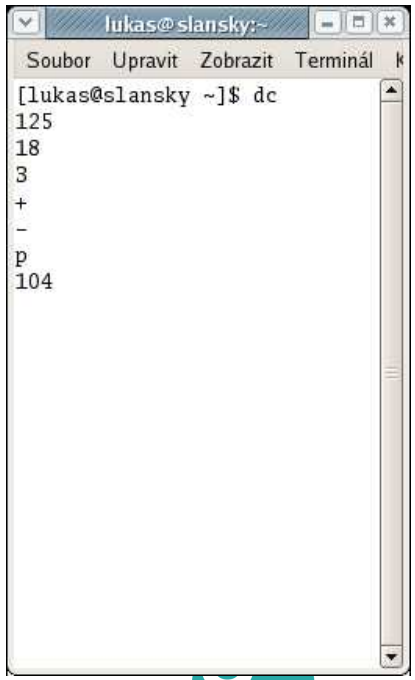
Example of Objects Composition

Design calculator that will work according to reverse polish notation. It can process basic math functions (+, -, *, /).



HP-55

TI-59



Unix programm dc

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition II.

- "Data are entered through keyboard and displayed on display."

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition II.

- "Data are entered through keyboard and displayed on display."
- "Pressing enter on keyboard stores input into stack."

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition II.

- "Data are entered through keyboard and displayed on display."
- "Pressing enter on keyboard stores input into stack."
- "Pressing operator (e.g. $+$) on keyboard retrieves data from stack and passes them to processor."

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition II.

- "Data are entered through keyboard and displayed on display."
- "Pressing enter on keyboard stores input into stack."
- "Pressing operator (e.g. $+$) on keyboard retrieves data from stack and passes them to processor."
- "Processor computes the result and pushes it into stack."

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition III.

Basic structure of program blocks:

- Computing part (simulation of simple processor).

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition III.

Basic structure of program blocks:

- Computing part (simulation of simple processor).
  - No input, no output!

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition III.

Basic structure of program blocks:

- Computing part (simulation of simple processor).
  - No input, no output!
  - As one of its' part is accumulator (stack) –

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition III.

Basic structure of program blocks:

- Computing part (simulation of simple processor).
  - No input, no output!
  - As one of its' part is accumulator (stack) – aggregation.

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Example of Objects Composition III.

Basic structure of program blocks:

- Computing part (simulation of simple processor).
  - No input, no output!
  - As one of its' part is accumulator (stack) – aggregation.
- Interactive part (display, keyboard).

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition III.

Basic structure of program blocks:

- Computing part (simulation of simple processor).
  - No input, no output!
  - As one of its' part is accumulator (stack) – aggregation.
- Interactive part (display, keyboard).
  - Just comunicates with user and computer.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition III.

Basic structure of program blocks:

- Computing part (simulation of simple processor).
  - No input, no output!
  - As one of its' part is accumulator (stack) – aggregation.
- Interactive part (display, keyboard).
  - Just comunicates with user and computer.
  - Only simple link to processor (it can be changed to other type) –

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Example of Objects Composition III.

Basic structure of program blocks:

- Computing part (simulation of simple processor).
  - No input, no output!
  - As one of its' part is accumulator (stack) – aggregation.
- Interactive part (display, keyboard).
  - Just comunicates with user and computer.
  - Only simple link to processor (it can be changed to other type) – association.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Consequences of Inheritance

- Descendant has all all features just like ascendant:

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Consequences of Inheritance

- Descendant has all all features just like ascendant:
  - it has all attributes like ascendat,
  - it has all methods like ascendant.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Consequences of Inheritance

- Descendant has all all features just like ascendant:
  - it has all attributes like ascendat,
  - it has all methods like ascendant.
- Descendant is capable of acting instead of ascendant.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Consequences of Inheritance

- Descendant has all all features just like ascendant:
  - it has all attributes like ascendat,
  - it has all methods like ascendant.
- Descendant is capable of acting instead of ascendant.
- Object from class of descendant can be assigned to object from class of ascendant.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Attributes Inheritance in Practice

```
type CPerson=object
       firstName:string;
       surname:string;
     end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
type CPerson=object
       firstName:string;
       surname:string;
     end;
     CPersonUPa=object(CPerson)
       faculty:string;
     end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
type CPerson=object
       firstName:string;
       surname:string;
     end;
     CPersonUPa=object(CPerson)
       faculty:string;
     end;
     CStudent=object(CPersonUPa)
       year:integer;
     end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
type CPerson=object
       firstName:string;
       surname:string;
     end;
     CPersonUPa=object(CPerson)
       faculty:string;
     end;
     CStudent=object(CPersonUPa)
       year:integer;
     end;
     CEmployee=object(CPersonUPa)
       office:string;
       fee:integer;
     end;
```

Attributes Inheritance in Practice

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
```

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
begin
  WriteLn(person.faculty);
end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
begin
  WriteLn(person.faculty);
end;
...
var p:CPerson;
    u:CPersonUPa;
    s:CStudent;
    e:CEmployee;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
begin
  WriteLn(person.faculty);
end;
...
var p:CPerson;
    u:CPersonUPa;
    s:CStudent;
    e:CEmployee;
...
printFaculty(u);
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
begin
  WriteLn(person.faculty);
end;
...
var p:CPerson;
    u:CPersonUPa;
    s:CStudent;
    e:CEmployee;
...
printFaculty(u);
printFaculty(s);
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
begin
  WriteLn(person.faculty);
end;
...
var p:CPerson;
    u:CPersonUPa;
    s:CStudent;
    e:CEmployee;
...
printFaculty(u);
printFaculty(s);
printFaculty(e);
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
begin
  WriteLn(person.faculty);
end;
...
var p:CPerson;
    u:CPersonUPa;
    s:CStudent;
    e:CEmployee;
...
printFaculty(u);
printFaculty(s);
printFaculty(e);
printFaculty(p);
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
begin
  WriteLn(person.faculty);
end;
...
var p:CPerson;
    u:CPersonUPa;
    s:CStudent;
    e:CEmployee;
...
printFaculty(u);
printFaculty(s);
printFaculty(e);
printFaculty(p);    Not possible!!!
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Inheritance in Practice

```
procedure printFaculty(person:CPersonUPa);
begin
  WriteLn(person.faculty);
end;
...
var p:CPerson;
    u:CPersonUPa;
    s:CStudent;
    e:CEmployee;
...
printFaculty(u);
printFaculty(s);
printFaculty(e);
printFaculty(p);    Not possible!!!   (there is no
                        "faculty" attribute)
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Placement in Memory

- Attributes in object are placed in order like in source code.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Placement in Memory

- Attributes in object are placed in order like in source code.
- Methods have no influence in placement of attributes.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Attributes Placement in Memory

- Attributes in object are placed in order like in source code.
- Methods have no influence in placement of attributes.
- When adding more attributes in descendant - new attributes are pleced behind original.

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Changing Methods During Inheritance

Method in inherited class can be redefined:

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Changing Methods During Inheritance

Method in inherited class can be redefined:

- Method must have the same name.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Changing Methods During Inheritance

Method in inherited class can be redefined:

- Method must have the same name.
- Parameters and return type can differ.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Changing Methods During Inheritance

Method in inherited class can be redefined:

- Method must have the same name.
- Parameters and return type can differ.
- For usage (and further inheritance) is used only redefined and the old one is overlayed.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Early and Late Binding

- Compiler knowns which function (better which method from which class) should be invoked during compilation.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Early and Late Binding

- Compiler knowns which function (better which method from which class) should be invoked during compilation.
  - *Early binding* = in the beginning (during compilation) are paired callings and placement of methods in memory.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Early and Late Binding

- Compiler knowns which function (better which method from which class) should be invoked during compilation.
  - *Early binding* = in the beginning (during compilation) are paired callings and placement of methods in memory.
  - Used in all previous cases of method calling.

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Early and Late Binding

- Compiler knowns which function (better which method from which class) should be invoked during compilation.
  - *Early binding* = in the beginning (during compilation) are paired callings and placement of methods in memory.
  - Used in all previous cases of method calling.
- Program decides which method should be invoked during run of program.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Early and Late Binding

- Compiler knowns which function (better which method from which class) should be invoked during compilation.
  - *Early binding* = in the beginning (during compilation) are paired callings and placement of methods in memory.
  - Used in all previous cases of method calling.
- Program decides which method should be invoked during run of program.
  - *Late binding* = method address (placement) is decided in time of invoking.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Early and Late Binding

- Compiler knowns which function (better which method from which class) should be invoked during compilation.
  - *Early binding* = in the beginning (during compilation) are paired callings and placement of methods in memory.
  - Used in all previous cases of method calling.
- Program decides which method should be invoked during run of program.
  - *Late binding* = method address (placement) is decided in time of invoking.
  - Will be used in some cases of redefined methods.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Early and Late Binding

- Compiler knowns which function (better which method from which class) should be invoked during compilation.
  - *Early binding* = in the beginning (during compilation) are paired callings and placement of methods in memory.
  - Used in all previous cases of method calling.
- Program decides which method should be invoked during run of program.
  - *Late binding* = method address (placement) is decided in time of invoking.
  - Will be used in some cases of redefined methods.
  - This case must be explicitly ordered.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Book Using Early Binding

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Book Using Early Binding

```
type EObjectType=(picture, paragraph);
type PPrintableObject=^CPrintableObject;
     CPrintableObject=object
       procedure print; {Abstract method}
       objectType:EObjectType;
     end;
     CPicture=object(CPrintableObject)
       procedure print;
     end;
     CParagraph=object(CPrintableObject)
       procedure print;
     end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Book Using Early Binding

```
type EObjectType=(picture, paragraph);
type PPrintableObject=^CPrintableObject;
     CPrintableObject=object
       procedure print; {Abstract method}
       objectType:EObjectType;
     end;
     CPicture=object(CPrintableObject)
       procedure print;
     end;
     CParagraph=object(CPrintableObject)
       procedure print;
     end;
     CBook=object
       printableObjects:array[...] of PPrintableObject;
       procedure print;
     end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Book Using Early Binding

```
procedure CBook.print;
var i:integer;
begin
  for i:=1 to printableObjectCount do
  begin
    case printableObjects[i].objectType of
      pictyre:  CPicture(printableObjects[i]^).print;
      paragraph: CParagraph(printableObjects[i]^).print;
    end;
  end;
end;
```

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Book Using Late Binding

```
procedure CBook.print;
var i:integer;
begin
  for i:=1 to printableObjectCount do
    printableObjects[i]^.print;
end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Book Using Late Binding

```
type EObjectType=(picture, paragraph);
type PPrintableObject=^CPrintableObject;
     CPrintableObject=object
       procedure print; {Abstract method}
       objectType:EObjectType;
     end;
     CPicture=object(CPrintableObject)
       procedure print;
     end;
     CParagraph=object(CPrintableObject)
       procedure print;
     end;
     CBook=object
       printableObjects:array[...] of PPrintableObjects;
       procedure print;
     end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Book Using Late Binding

```
type EObjectType=(picture, paragraph);
type PPrintableObject=^CPrintableObject;
     CPrintableObject=object
       procedure print;virtual; {Abstract method}
       objectType:EObjectType;
     end;
     CPicture=object(CPrintableObject)
       procedure print;virtual;
     end;
     CParagraph=object(CPrintableObject)
       procedure print;virtual;
     end;
     CBook=object
       printableObjects:array[...] of PPrintableObjects;
       procedure print;
     end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Book Using Late Binding

```
type PPrintableObject=^CPrintableObject;
     CPrintableObject=object
       procedure print;virtual; {Abstract method}

     end;
     CPicture=object(CPrintableObject)
       procedure print;virtual;
     end;
     CParagraph=object(CPrintableObject)
       procedure print;virtual;
     end;
     CBook=object
       printableObjects:array[...] of PPrintableObjects;
       procedure print;
     end;
```

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Virtual methods

- The VMT (Virtual Methods Table – tabulka virtuálních metod) is created when defining virtual method.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Virtual methods

- The VMT (Virtual Methods Table – tabulka virtuálních metod) is created when defining virtual method.
  - VMT contains pointers to methods that should be invoked in program intends calling one of virtual methods.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Virtual methods

- The VMT (Virtual Methods Table – tabulka virtuálních metod) is created when defining virtual method.
  - VMT contains pointers to methods that should be invoked in program intends calling one of virtual methods.
  - Each object carries unique VMT as one of the attributes.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Virtual methods

- The VMT (Virtual Methods Table – tabulka virtuálních metod) is created when defining virtual method.
  - VMT contains pointers to methods that should be invoked in program intends calling one of virtual methods.
  - Each object carries unique VMT as one of the attributes.
- VMT is using during calling virtual methods.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Using Virtual Methods

Some preconditions must be fulfilled for using virtual methods:

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Using Virtual Methods

Some preconditions must be fulfilled for using virtual methods:

- Virtual method must have the same declaration (i.e. same attributes and return type).

Univerzita
Pardubice
Fakulta elektrotechniky
a informatiky

Using Virtual Methods

Some preconditions must be fulfilled for using virtual methods:

- Virtual method must have the same declaration (i.e. same attributes and return type).
- Class must contain special method called *constructor*
  E.g.: `constructor init;`

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Using Virtual Methods

Some preconditions must be fulfilled for using virtual methods:

- Virtual method must have the same declaration (i.e. same attributes and return type).
- Class must contain special method called *constructor*
  E.g.: `constructor init;`
  - Constructor fills VMT before running its' code.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Using Virtual Methods

Some preconditions must be fulfilled for using virtual methods:

- Virtual method must have the same declaration (i.e. same attributes and return type).
- Class must contain special method called *constructor*
  E.g.: `constructor init;`
  - Constructor fills VMT before running its' code.
  - Constructor must be called before first usage of the object.

Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Using Virtual Methods

Some preconditions must be fulfilled for using virtual methods:

- Virtual method must have the same declaration (i.e. same attributes and return type).
- Class must contain special method called *constructor*
  E.g.: `constructor init;`
  - Constructor fills VMT before running its' code.
  - Constructor must be called before first usage of the object.
  - If you use virtual method before calling the constructor, the consequences are inpredictable (program mostly crashes).

Univerzita Pardubice
Fakulta elektrotechniky a informatiky