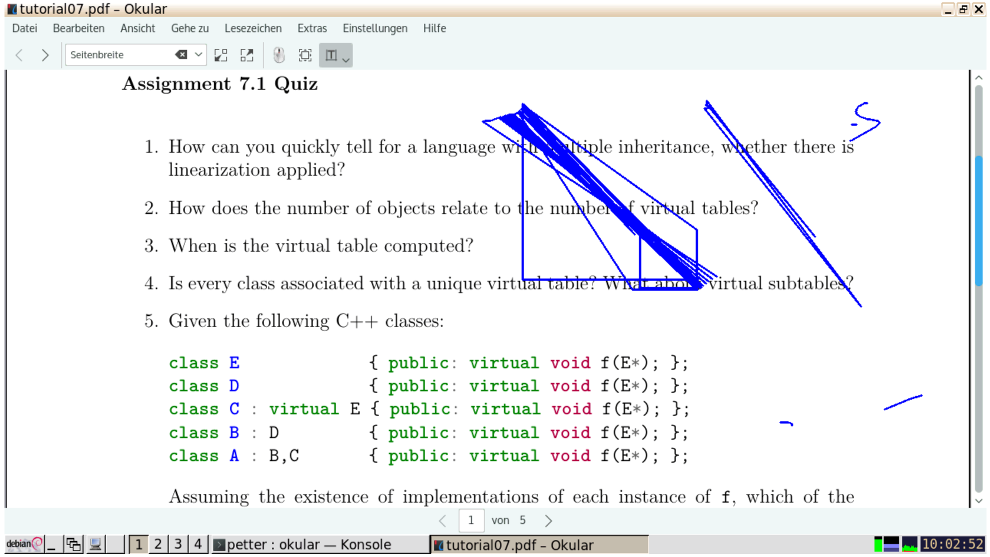
Script **generated by TTT**

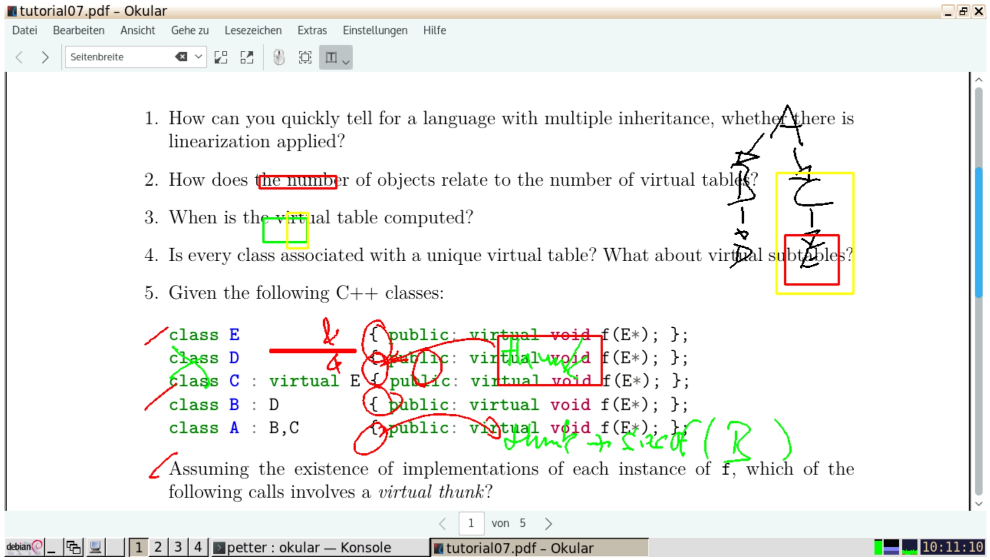
Title: Petter: Programmiersprachen_Uebung (23.12.2016)

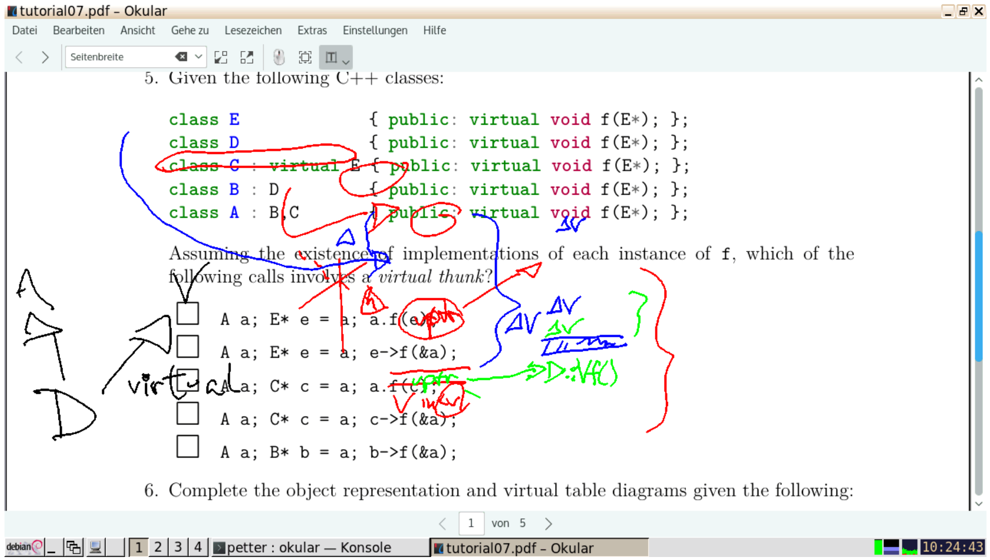
Date: Fri Dec 23 10:02:52 CET 2016

Duration: 123:08 min

Pages: 25







7. Consider the following code (note: shared base class):

```
class A { };
class B : public virtual A { };
class C : public virtual A { };
class D : public B, public C { };
...
C c; A* a = &c; (C*)a;
```

Is the cast correct?

Assignment 7.2 Linearization

Consider the following classes: `A(B,C) B(D,E) C(F,G) D(G) E(F)`
Give the linearization order of the following methods:

1. LPDFS with Duplicate Cancellation

7. Consider the following code (note: shared base class):

```
class A { };
class B : public virtual A { };
class C : public virtual A { };
class D : public B, public C { };
...
C c; A* a = &c; (C*)a;
```

Is the cast correct?

Assignment 7.2 Linearization

Consider the following classes: `A(B,C) B(D,E) C(F,G) D(G) E(F)`
Give the linearization order of the following methods:

1. LPDFS with Duplicate Cancellation

Assignment 7.2 Linearization

Consider the following classes: `A(B,C) B(D,E) C(F,G) D(G) E(F)`
Give the linearization order of the following methods:

1. LPDFS with Duplicate Cancellation
2. Reverse Postorder Rightmost DFS
3. C3

A

Assignment 7.3 Multiple Inheritance – straight from the Exam 2015

Provide a C++ class structure and a main function, which failes to compile, due to multiple inheritance causing

1. ... an ambiguously resolvable call expression
2. ... ambiguous casting target types

A(B, C)
B(D, E)
C(F, G)
D(G)
E(F)

L(F) = F
L(G) = G
L(D) = D · G
L(E) = E · F
L(C) = C · ⊔(F, G, F · G) = C · F · G
L(B) = B · ⊔(DG, EF, D · E) = BDGEF
L(A) = A · ⊔(BDGEF, CFG, BC)

1. ABDGEF fail G<->F
2. GECFDBA
2 ABDECFG

Assignment 7.2 Linearization

Consider the following classes: `A(B,C) B(D,E) C(F,G) D(G) E(F)`
Give the linearization order of the following methods:

1. LPDFS with Duplicate Cancellation
2. Reverse Postorder Rightmost DFS
3. C3

Assignment 7.3 Multiple Inheritance – straight from the Exam 2015

Provide a C++ class structure and a main function which failes to compile, due to multiple inheritance causing

1. ... an ambiguously resolvable call expression
2. ... ambiguous casting target types

Af Cf
B
A
ARvi
C
A* a = (A*) C;
C

Assignment 7.4 Multiple Inheritance – straight from the Exam 2014

This C++ code defines a few classes:

```
class A {
public:
  int a;
  virtual int f(int);
  virtual int g(int);
};
class B : public A {
public:
  int b;
  int f(int);
  virtual int h(int);
```

This is the Virtual Table for class D:

```
Entry| Value
   0 | vbase_offset (40)
   1 | offset_to_top (0)
   2 | D RTTI
   3 | int D::f(int)
   4 | offset_to_top (-16)
   5 | D RTTI
   6 | int D::f(int)
```



```
public:
  int a;
  virtual int f(int);
  virtual int g(int);
};
class B : public A {
public:
  int b;
  int f(int);
  virtual int h(int);
};
class C : virtual public A {
public:
  int c;
  int f(int);
};
class D: public C, B {
public:
  int d;
  int f(int);
};
```

This is the Virtual Table for class D:

```
Entry| Value
   0 | vbase_offset (40)
   1 | offset_to_top (0)
   2 | D RTTI
   3 | int D::f(int)
   4 | offset_to_top (-16)
   5 | D RTTI
   6 | int D::f(int)
   7 | int A::g(int)
   8 | int B::h(int)
   9 | vcall_offset (0)
  10 | vcall_offset (-40)
  11 | offset_to_top (-40)
  12 | D RTTI
  13 | int D::f(int)
  14 | int A::g(int)
```

1. Draw the layout for a class D object memory representation!


```
class A {
public:
  int a;
  virtual int f(int);
  virtual int g(int);
};
class B : public A {
public:
  int b;
  int f(int);
  virtual int h(int);
};
class C : virtual public A {
public:
  int c;
  int f(int);
};
class D: public C, B {
public:
  int d;
  int f(int);
};
```

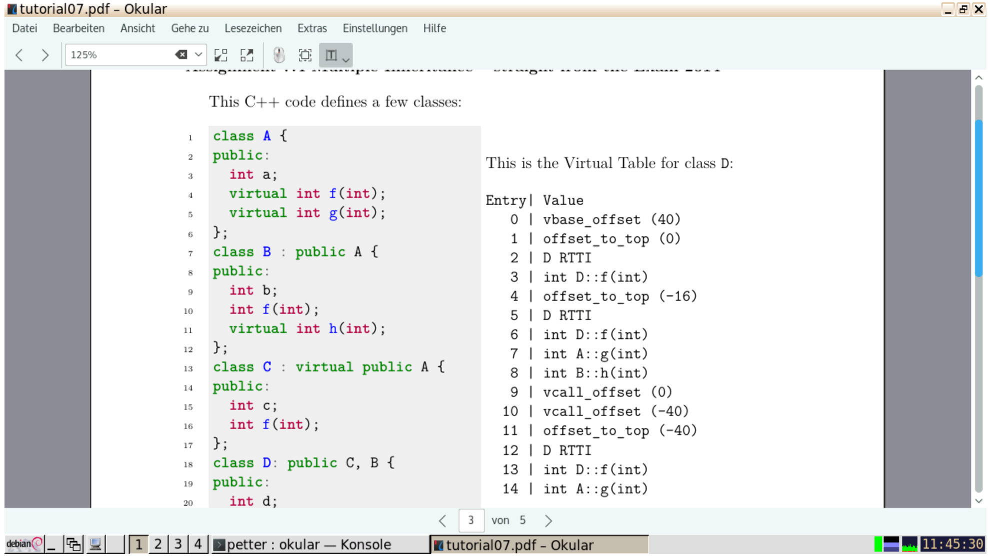
This is the Virtual Table for class D:

```
Entry| Value
   0 | vbase_offset (40)
   1 | offset_to_top (0)
   2 | D RTTI
   3 | int D::f(int)
   4 | offset_to_top (-16)
   5 | D RTTI
   6 | int D::f(int)
   7 | int A::g(int)
   8 | int B::h(int)
   9 | vcall_offset (0)
  10 | vcall_offset (-40)
  11 | offset_to_top (-40)
  12 | D RTTI
  13 | int D::f(int)
  14 | int A::g(int)
```


A vptr
int a
int b

C vptr
int c
A vptr

C vptr 3
int c
A vptr 6
int a
int b
D int d
A vptr = 13
int a



tutorial07.pdf - Okular

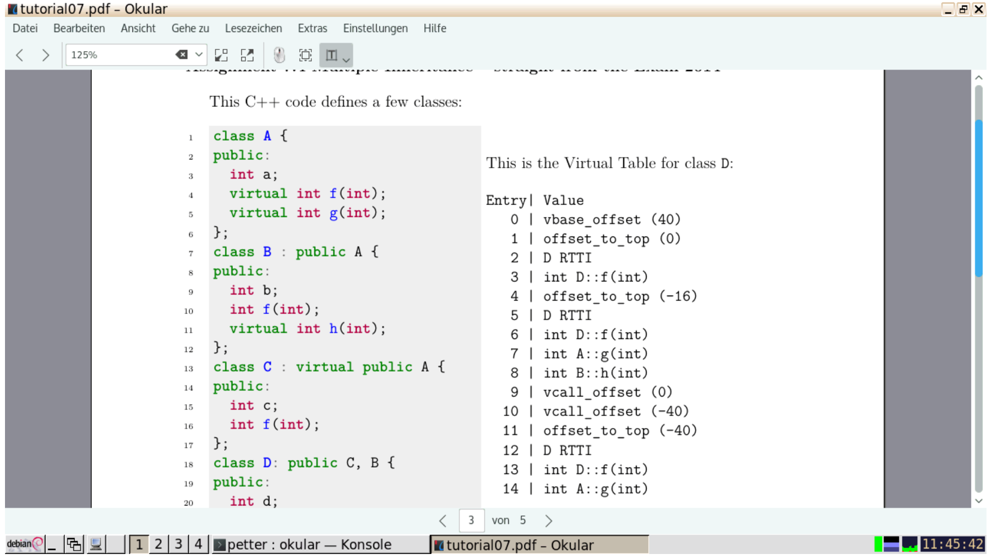
This C++ code defines a few classes:

```
class A {
public:
  int a;
  virtual int f(int);
  virtual int g(int);
};
class B : public A {
public:
  int b;
  int f(int);
  virtual int h(int);
};
class C : virtual public A {
public:
  int c;
  int f(int);
};
class D: public C, B {
public:
  int d;
```

This is the Virtual Table for class D:

```
Entry| Value
   0 | vbase_offset (40)
   1 | offset_to_top (0)
   2 | D RTTI
   3 | int D::f(int)
   4 | offset_to_top (-16)
   5 | D RTTI
   6 | int D::f(int)
   7 | int A::g(int)
   8 | int B::h(int)
   9 | vcall_offset (0)
  10 | vcall_offset (-40)
  11 | offset_to_top (-40)
  12 | D RTTI
  13 | int D::f(int)
  14 | int A::g(int)
```

3 von 5



tutorial07.pdf - Okular

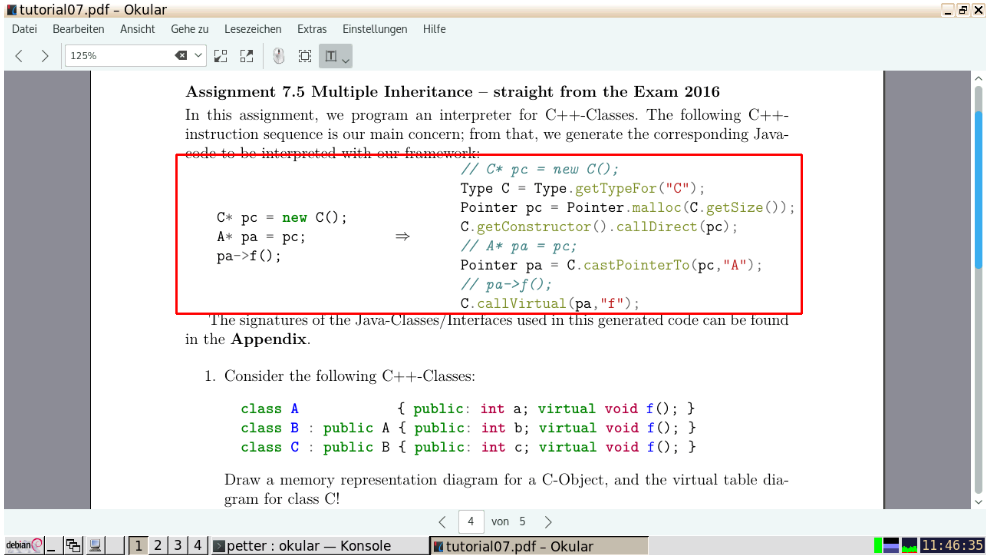
This C++ code defines a few classes:

```
class A {
public:
  int a;
  virtual int f(int);
  virtual int g(int);
};
class B : public A {
public:
  int b;
  int f(int);
  virtual int h(int);
};
class C : virtual public A {
public:
  int c;
  int f(int);
};
class D: public C, B {
public:
  int d;
```

This is the Virtual Table for class D:

```
Entry| Value
   0 | vbase_offset (40)
   1 | offset_to_top (0)
   2 | D RTTI
   3 | int D::f(int)
   4 | offset_to_top (-16)
   5 | D RTTI
   6 | int D::f(int)
   7 | int A::g(int)
   8 | int B::h(int)
   9 | vcall_offset (0)
  10 | vcall_offset (-40)
  11 | offset_to_top (-40)
  12 | D RTTI
  13 | int D::f(int)
  14 | int A::g(int)
```

3 von 5



tutorial07.pdf - Okular

Assignment 7.5 Multiple Inheritance – straight from the Exam 2016

In this assignment, we program an interpreter for C++-Classes. The following C++-instruction sequence is our main concern; from that, we generate the corresponding Java-code to be interpreted with our framework:

```
C* pc = new C();
A* pa = pc;
pa->f();
```

⇒

```
// C* pc = new C();
Type C = Type.getTypeFor("C");
Pointer pc = Pointer.malloc(C.getSize());
C.getConstructor().callDirect(pc);
// A* pa = pc;
Pointer pa = C.castPointerTo(pc,"A");
// pa->f();
C.callVirtual(pa,"f");
```

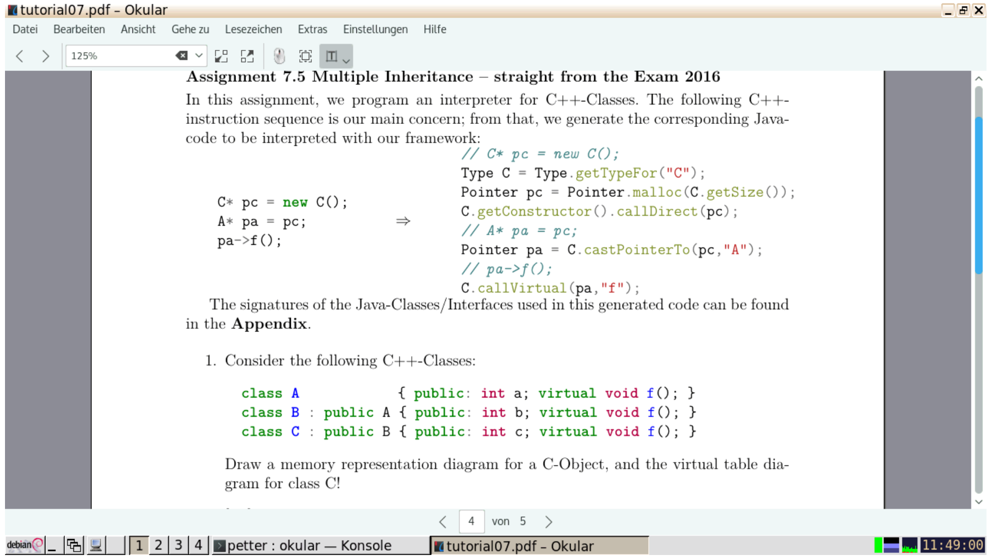
The signatures of the Java-Classes/Interfaces used in this generated code can be found in the **Appendix**.

1. Consider the following C++-Classes:

```
class A             { public: int a; virtual void f(); }
class B : public A { public: int b; virtual void f(); }
class C : public B { public: int c; virtual void f(); }
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

4 von 5



tutorial07.pdf - Okular

Assignment 7.5 Multiple Inheritance – straight from the Exam 2016

In this assignment, we program an interpreter for C++-Classes. The following C++-instruction sequence is our main concern; from that, we generate the corresponding Java-code to be interpreted with our framework:

```
C* pc = new C();
A* pa = pc;
pa->f();
```

⇒

```
// C* pc = new C();
Type C = Type.getTypeFor("C");
Pointer pc = Pointer.malloc(C.getSize());
C.getConstructor().callDirect(pc);
// A* pa = pc;
Pointer pa = C.castPointerTo(pc,"A");
// pa->f();
C.callVirtual(pa,"f");
```

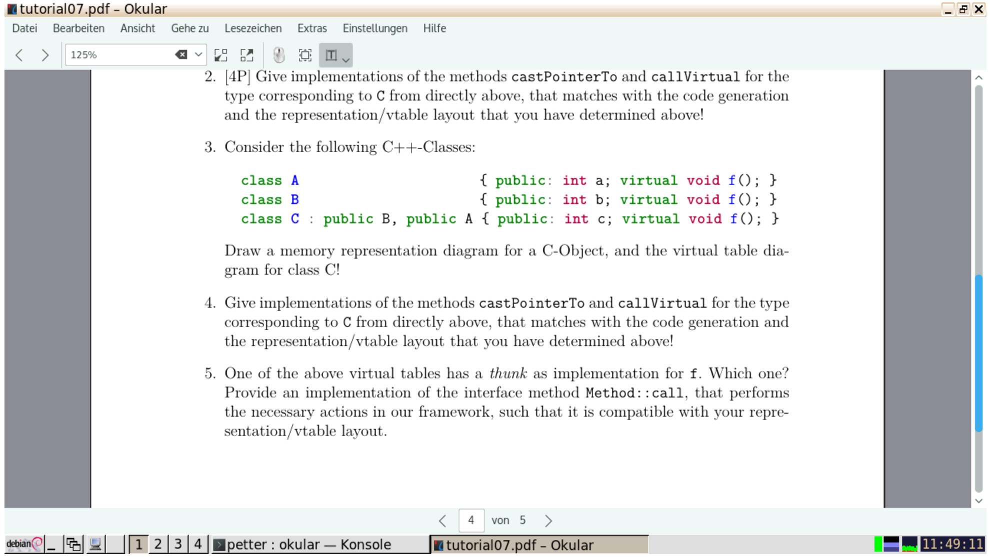
The signatures of the Java-Classes/Interfaces used in this generated code can be found in the **Appendix**.

1. Consider the following C++-Classes:

```
class A             { public: int a; virtual void f(); }
class B : public A { public: int b; virtual void f(); }
class C : public B { public: int c; virtual void f(); }
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

4 von 5



2. [4P] Give implementations of the methods `castPointerTo` and `callVirtual` for the type corresponding to `C` from directly above, that matches with the code generation and the representation/vtable layout that you have determined above!

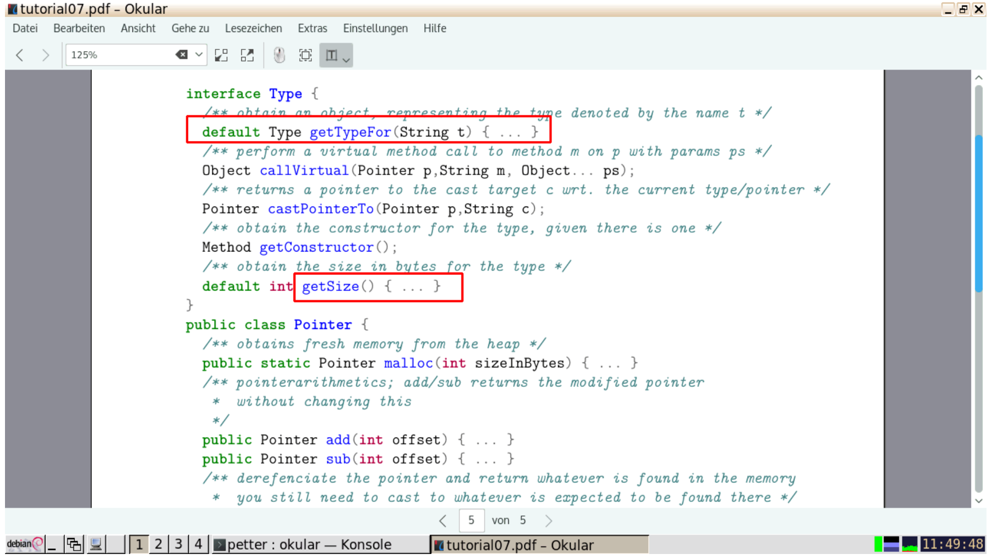
3. Consider the following C++-Classes:

```
class A                      { public: int a; virtual void f(); }
class B                      { public: int b; virtual void f(); }
class C : public B, public A { public: int c; virtual void f(); }
```

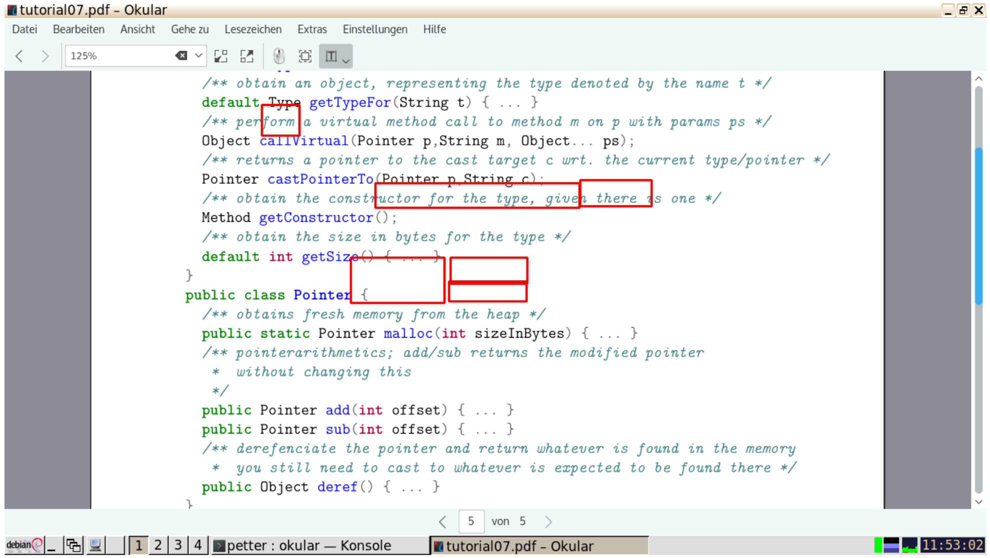
Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

4. Give implementations of the methods `castPointerTo` and `callVirtual` for the type corresponding to `C` from directly above, that matches with the code generation and the representation/vtable layout that you have determined above!

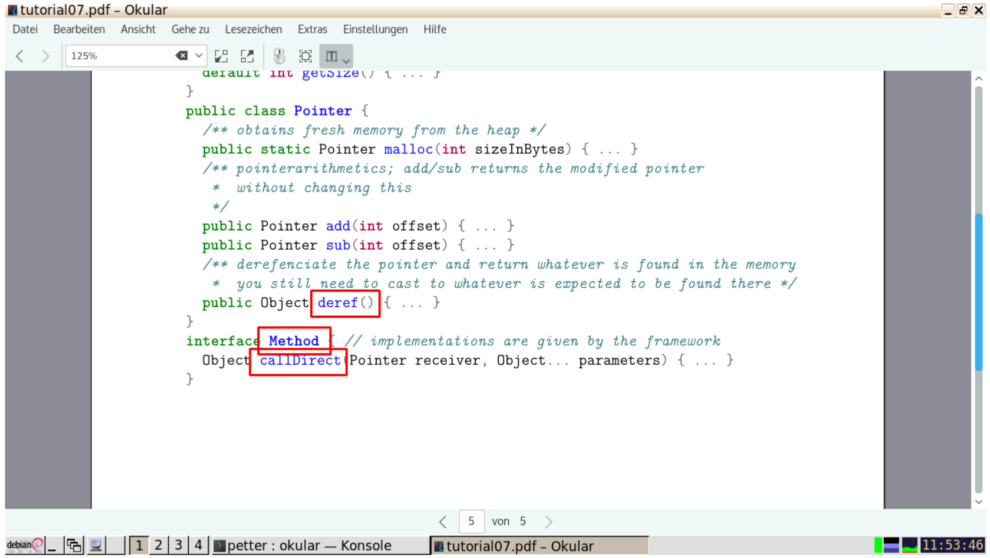
5. One of the above virtual tables has a *thunk* as implementation for `f`. Which one? Provide an implementation of the interface method `Method::call`, that performs the necessary actions in our framework, such that it is compatible with your representation/vtable layout.



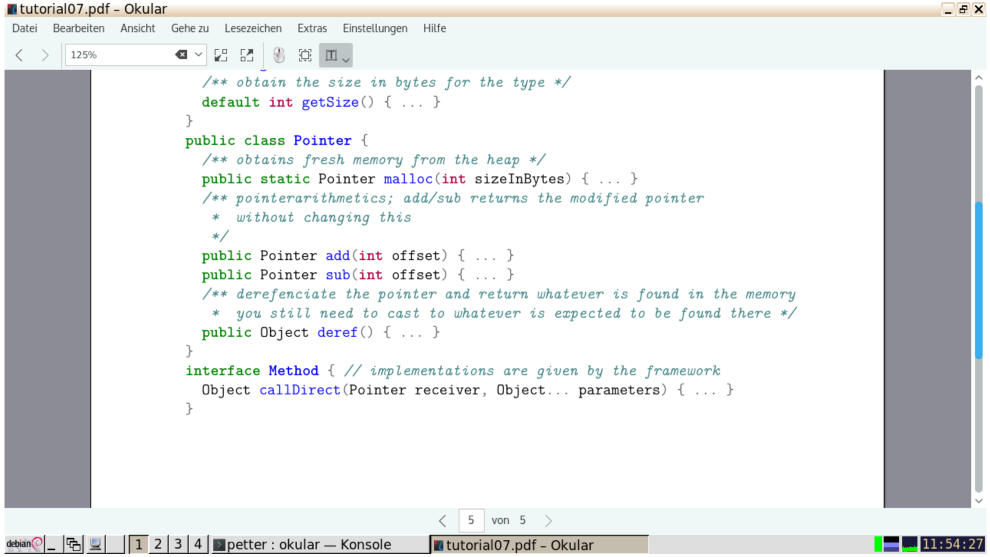

```
interface Type {
  /** obtain an object, representing the type denoted by the name t */
  default Type getTypeFor(String t) { ... }
  /** perform a virtual method call to method m on p with params ps */
  Object callVirtual(Pointer p,String m, Object... ps);
  /** returns a pointer to the cast target c wrt. the current type/pointer */
  Pointer castPointerTo(Pointer p,String c);
  /** obtain the constructor for the type, given there is one */
  Method getConstructor();
  /** obtain the size in bytes for the type */
  default int getSize() { ... }
}
public class Pointer {
  /** obtains fresh memory from the heap */
  public static Pointer malloc(int sizeInBytes) { ... }
  /** pointerarithmetics; add/sub returns the modified pointer
   *  without changing this
   */
  public Pointer add(int offset) { ... }
  public Pointer sub(int offset) { ... }
  /** dereferenciate the pointer and return whatever is found in the memory
   *  you still need to cast to whatever is expected to be found there */
```




```
  /** obtain an object, representing the type denoted by the name t */
  default Type getTypeFor(String t) { ... }
  /** perform a virtual method call to method m on p with params ps */
  Object callVirtual(Pointer p,String m, Object... ps);
  /** returns a pointer to the cast target c wrt. the current type/pointer */
  Pointer castPointerTo(Pointer p,String c);
  /** obtain the constructor for the type, given there is one */
  Method getConstructor();
  /** obtain the size in bytes for the type */
  default int getSize() { ... }
}
public class Pointer {
  /** obtains fresh memory from the heap */
  public static Pointer malloc(int sizeInBytes) { ... }
  /** pointerarithmetics; add/sub returns the modified pointer
   *  without changing this
   */
  public Pointer add(int offset) { ... }
  public Pointer sub(int offset) { ... }
  /** dereferenciate the pointer and return whatever is found in the memory
   *  you still need to cast to whatever is expected to be found there */
  public Object deref() { ... }
}
```




```
  default int getSize() { ... }
}
public class Pointer {
  /** obtains fresh memory from the heap */
  public static Pointer malloc(int sizeInBytes) { ... }
  /** pointerarithmetics; add/sub returns the modified pointer
   *  without changing this
   */
  public Pointer add(int offset) { ... }
  public Pointer sub(int offset) { ... }
  /** dereferenciate the pointer and return whatever is found in the memory
   *  you still need to cast to whatever is expected to be found there */
  public Object deref() { ... }
}
interface Method { // implementations are given by the framework
  Object callDirect(Pointer receiver, Object... parameters) { ... }
}
```

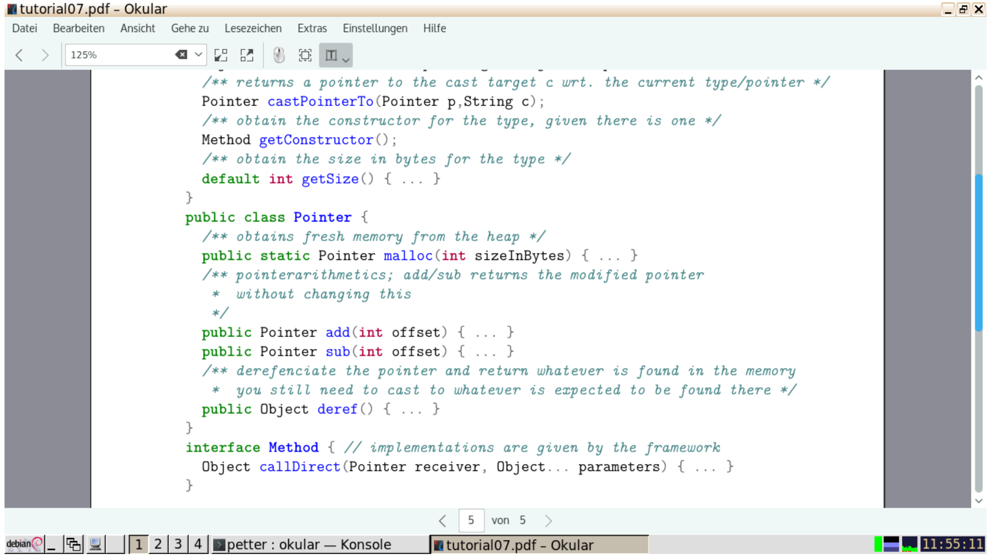


tutorial07.pdf – Okular

```
    /** obtain the size in bytes for the type */
    default int getSize() { ... }
  }
  public class Pointer {
    /** obtains fresh memory from the heap */
    public static Pointer malloc(int sizeInBytes) { ... }
    /** pointerarithmetics; add/sub returns the modified pointer
     *  without changing this
     */
    public Pointer add(int offset) { ... }
    public Pointer sub(int offset) { ... }
    /** derefenciate the pointer and return whatever is found in the memory
     *  you still need to cast to whatever is expected to be found there */
    public Object deref() { ... }
  }
  interface Method { // implementations are given by the framework
    Object callDirect(Pointer receiver, Object... parameters) { ... }
  }
```

5 von 5

petter : okular — Konsole | tutorial07.pdf – Okular | 11:54:27

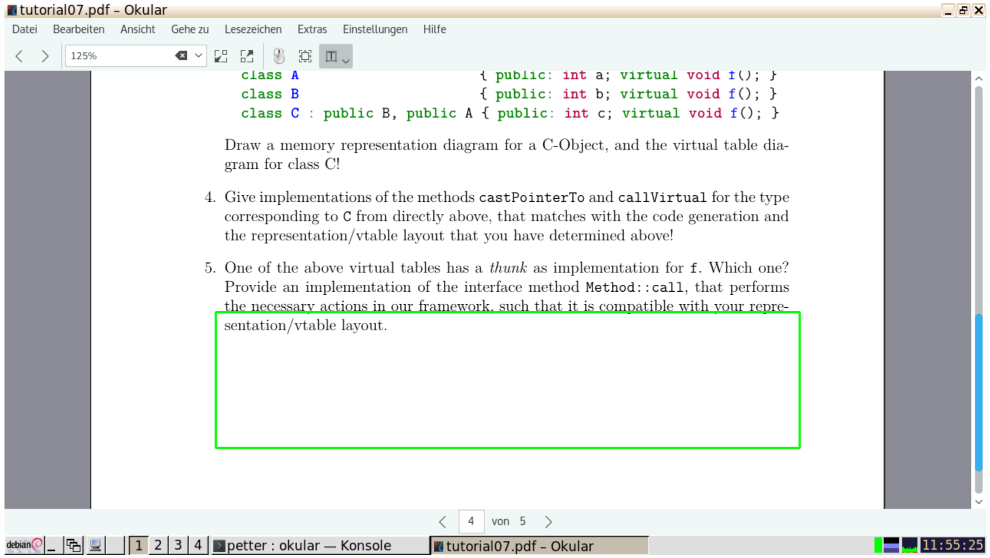


tutorial07.pdf – Okular

```
    /** returns a pointer to the cast target c wrt. the current type/pointer */
    Pointer castPointerTo(Pointer p,String c);
    /** obtain the constructor for the type, given there is one */
    Method getConstructor();
    /** obtain the size in bytes for the type */
    default int getSize() { ... }
  }
  public class Pointer {
    /** obtains fresh memory from the heap */
    public static Pointer malloc(int sizeInBytes) { ... }
    /** pointerarithmetics; add/sub returns the modified pointer
     *  without changing this
     */
    public Pointer add(int offset) { ... }
    public Pointer sub(int offset) { ... }
    /** derefenciate the pointer and return whatever is found in the memory
     *  you still need to cast to whatever is expected to be found there */
    public Object deref() { ... }
  }
  interface Method { // implementations are given by the framework
    Object callDirect(Pointer receiver, Object... parameters) { ... }
  }
```

5 von 5

petter : okular — Konsole | tutorial07.pdf – Okular | 11:55:11



tutorial07.pdf – Okular

```
class A                       { public: int a; virtual void f(); }
class B                       { public: int b; virtual void f(); }
class C : public B, public A { public: int c; virtual void f(); }
```

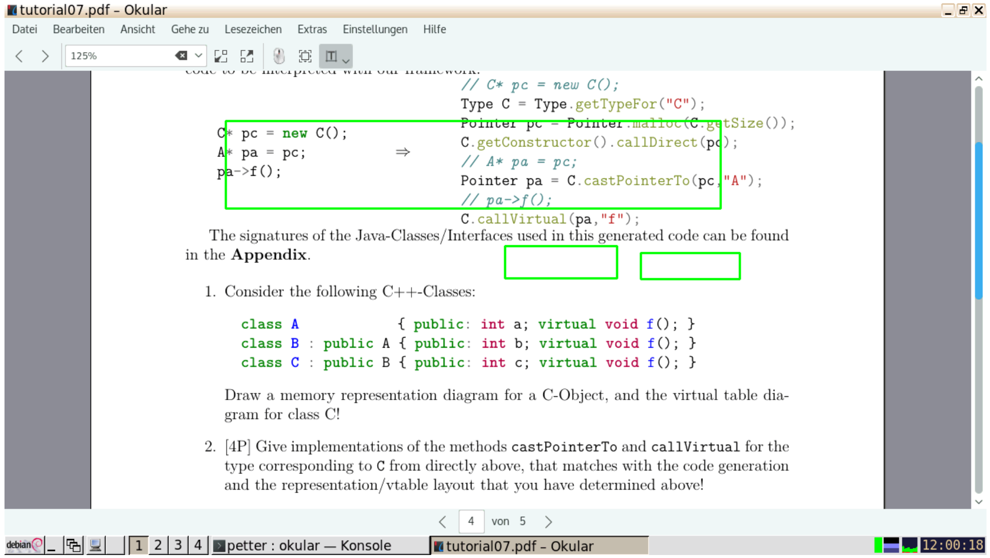
Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

4. Give implementations of the methods `castPointerTo` and `callVirtual` for the type corresponding to `C` from directly above, that matches with the code generation and the representation/vtable layout that you have determined above!

5. One of the above virtual tables has a *thunk* as implementation for `f`. Which one? Provide an implementation of the interface method `Method::call`, that performs the necessary actions in our framework, such that it is compatible with your representation/vtable layout.

4 von 5

petter : okular — Konsole | tutorial07.pdf – Okular | 11:55:25



tutorial07.pdf – Okular

```
                          // C* pc = new C();
                          Type C = Type.getTypeFor("C");
                          Pointer pc = Pointer.malloc(C.getSize());
C* pc = new C();          C.getConstructor().callDirect(pc);
A* pa = pc;          ⇒    // A* pa = pc;
pa->f();                  Pointer pa = C.castPointerTo(pc,"A");
                          // pa->f();
                          C.callVirtual(pa,"f");
```

The signatures of the Java-Classes/Interfaces used in this generated code can be found in the **Appendix**.

1. Consider the following C++-Classes:

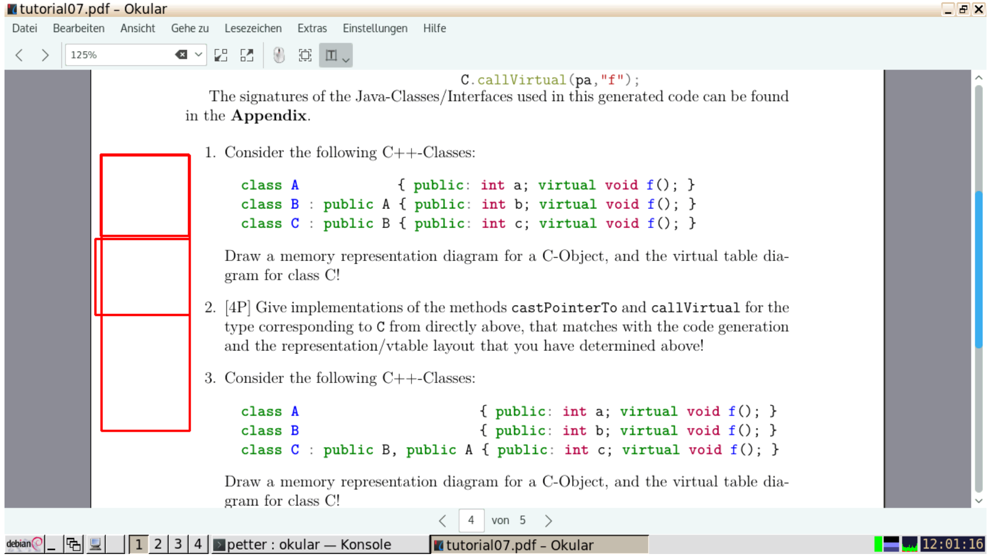
```
class A             { public: int a; virtual void f(); }
class B : public A { public: int b; virtual void f(); }
class C : public B { public: int c; virtual void f(); }
```

Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

2. [4P] Give implementations of the methods `castPointerTo` and `callVirtual` for the type corresponding to `C` from directly above, that matches with the code generation and the representation/vtable layout that you have determined above!

4 von 5

petter : okular — Konsole | tutorial07.pdf – Okular | 12:00:18



tutorial07.pdf – Okular

Datei Bearbeiten Ansicht Gehe zu Lesezeichen Extras Einstellungen Hilfe

125%

```
C.callVirtual(pa,"f");
```

The signatures of the Java-Classes/Interfaces used in this generated code can be found in the **Appendix**.

1. Consider the following C++-Classes:

```
class A             { public: int a; virtual void f(); }
class B : public A { public: int b; virtual void f(); }
class C : public B { public: int c; virtual void f(); }
```

   Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

2. [4P] Give implementations of the methods `castPointerTo` and `callVirtual` for the type corresponding to `C` from directly above, that matches with the code generation and the representation/vtable layout that you have determined above!

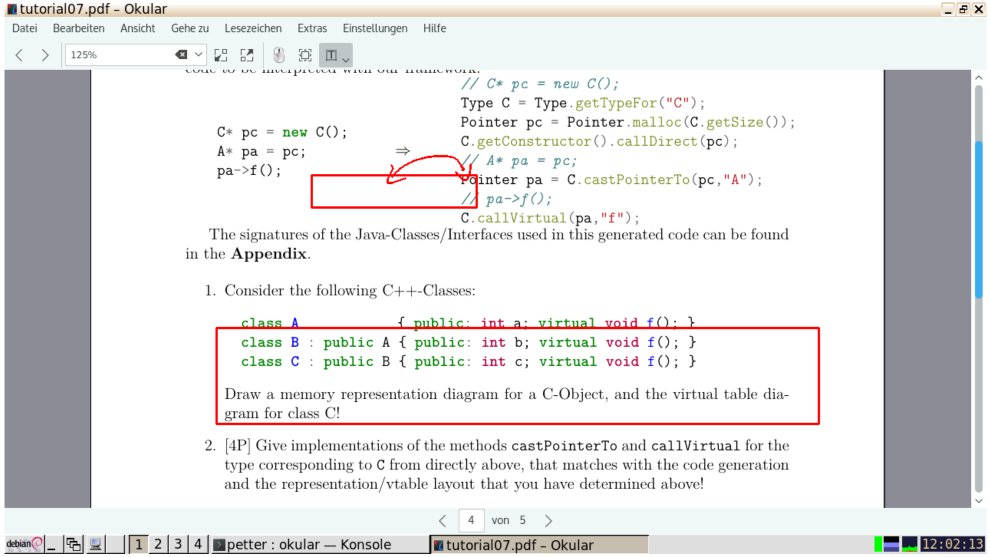
3. Consider the following C++-Classes:

```
class A                         { public: int a; virtual void f(); }
class B                         { public: int b; virtual void f(); }
class C : public B, public A { public: int c; virtual void f(); }
```

   Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

4 von 5

debian 1 2 3 4 petter : okular — Konsole tutorial07.pdf – Okular 12:01:16



tutorial07.pdf – Okular

Datei Bearbeiten Ansicht Gehe zu Lesezeichen Extras Einstellungen Hilfe

125%

```
C* pc = new C();            // C* pc = new C();
A* pa = pc;                 Type C = Type.getTypeFor("C");
pa->f();            ⇒       Pointer pc = Pointer.malloc(C.getSize());
                            C.getConstructor().callDirect(pc);
                            // A* pa = pc;
                            Pointer pa = C.castPointerTo(pc,"A");
                            // pa->f();
                            C.callVirtual(pa,"f");
```

The signatures of the Java-Classes/Interfaces used in this generated code can be found in the **Appendix**.

1. Consider the following C++-Classes:

```
class A             { public: int a; virtual void f(); }
class B : public A { public: int b; virtual void f(); }
class C : public B { public: int c; virtual void f(); }
```

   Draw a memory representation diagram for a C-Object, and the virtual table diagram for class C!

2. [4P] Give implementations of the methods `castPointerTo` and `callVirtual` for the type corresponding to `C` from directly above, that matches with the code generation and the representation/vtable layout that you have determined above!

4 von 5

debian 1 2 3 4 petter : okular — Konsole tutorial07.pdf – Okular 12:02:13