# Konzepte objektorientierter Programmierung

**Prof. Dr. Peter Müller**

**Werner Dietl**

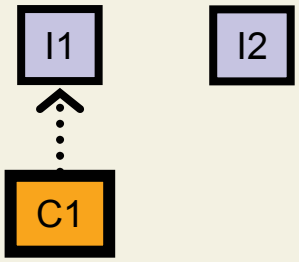
Software Component Technology

Exercises 3: Inheritance and Subtyping in Sather

Wintersemester 03/04

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Homework – Exercise 1



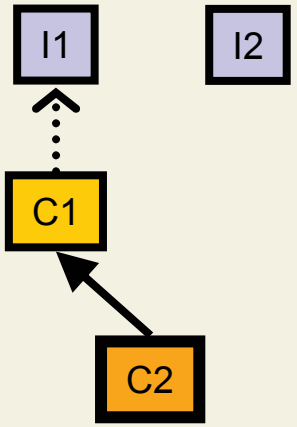

```
interface I1 {}
interface I2 {}
final class C1 implements I1 {}

public class Ex1 {
    public static void main( String[] args ) {
        C1 c1 = new C1();
        I2 i2 = (I2) c1;
    }
}
```

- Compilation Error: inconvertible types



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Homework – Exercise 2




```
interface I1 {}
interface I2 {}
class C1 implements I1 {}
final class C2 extends C1 {}

public class Ex2 {
  public static void main( String[] args ) {
      C1 c1 = new C2();
      I2 i2 = (I2) c1;
  }
}
```

- Runtime Error: java.lang.ClassCastException



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Homework – Exercise 3

```
class C1 {
    int age = 99;
    int getAge() {
      return age; }
}

class C2 extends C1 {
    int age = 22;
    int getAge() {
      return age; }
}
```

- Output:

```
C1.age:          99
C1.getAge:       99
C2.age:          22
C2.getAge:       22
C1=C2.age:       99
C1=C2.getAge:    22
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Assertions (since Java 1.4)

- Two versions of assertions:
```
assert Expression1 ;
assert Expression1 : Expression2 ;
```
- Java evaluates `Expression1` and if it is false throws an `AssertionError` with the value of `Expression2` as detail message, if present.
- Compile like this:
```
javac -source 1.4 MyClass.java
```
- At runtime enable/disable assertions selectively per package or class:
```
java –enableassertions:pack.age... Test
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Assertions for Pre- and Post-Conditions

- Assertion statements can be used to simulate Design-by-Contract in a simple way
- For public methods use if-statements to check the input, because assertions can be turned off
- But for internal methods you can use asserts to make sure the preconditions are met
- Assertions at the end of a method can be used to establish postconditions
- Consistently adding checks to the postconditions can establish a class invariant

# Assertions Example

- Internal function for updating an attribute:

```
private int attr;
void setAttr( int newAttr ) {
        assert newAttr > 0 :
                "New attribute value <= 0!";
        attr = newAttr;
}
```

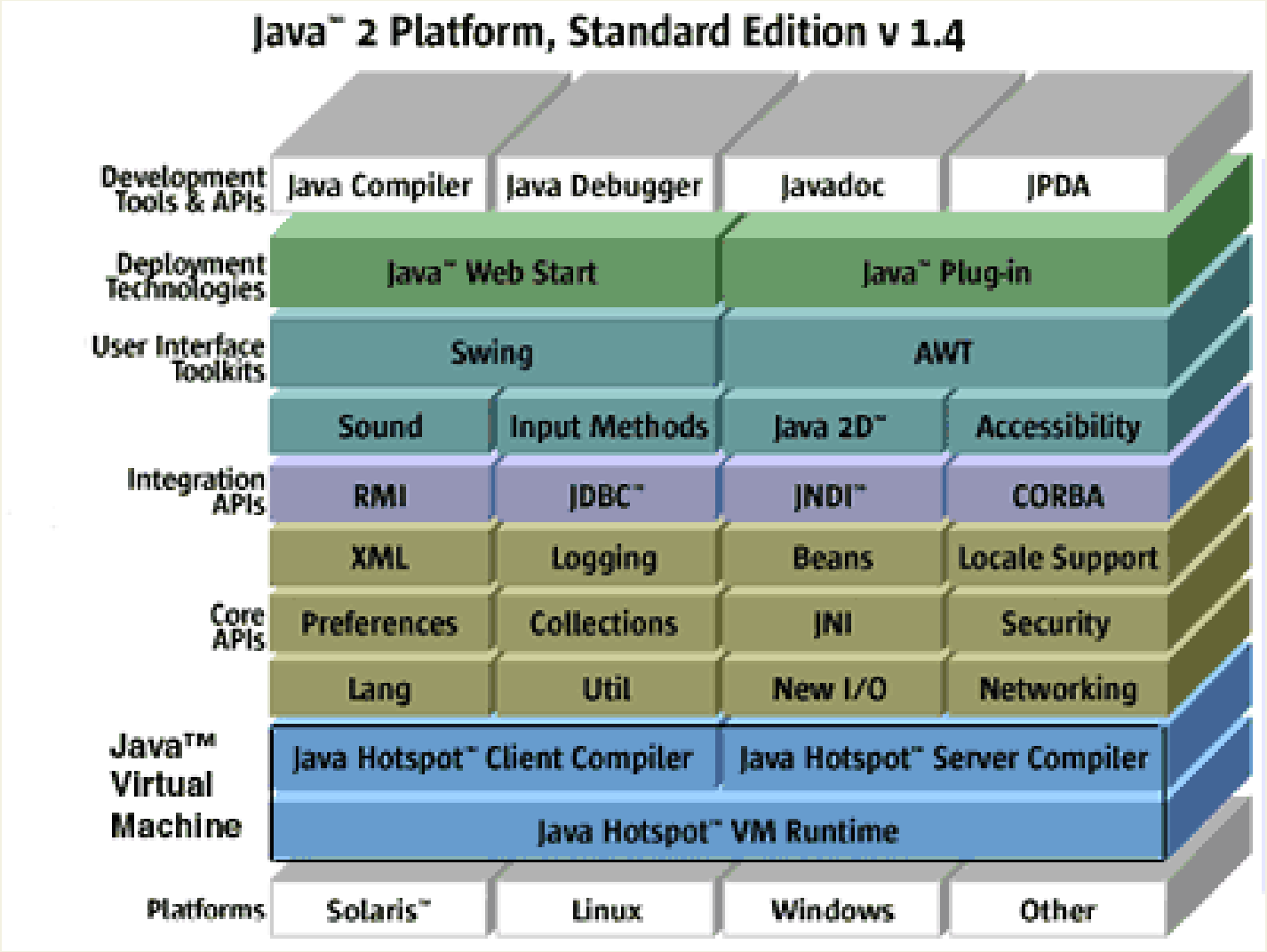
- An invalid call at runtime results in:

```
Exception in thread "main" java.lang.AssertionError:
        New attribute value <= 0!
    at Test9.setAttr(Test9.java:8)
    at Test9.main(Test9.java:23)
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Java 2 Platform, Standard Edition






ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# What's to come in Java 1.5

- Generics:

```
List<String> words =
        new ArrayList<String>();
```

- Enhanced `for`-loop:

```
void cancelAll(Collection c){
   for (Object o : c)
        ((TimerTask)o).cancel();
}
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# What's to come in Java 1.5

- Autoboxing/unboxing

```
public class Freq {
   public static void main(String args[]) {
      Map<String, Integer> m =
         new TreeMap<String, Integer>();
      for(String word: args)
         m.put(word, m.get(word) + 1);
      System.out.println(m);
   }
}
```

- Other things: typesafe enums, static import, metadata, …



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Sather Features

- object-oriented dynamic binding,
- statically-checked strong typing,
- separate implementation and type inheritance,
- multiple inheritance,
- garbage collection,
- iteration abstraction,
- parameterized classes,
- higher-order routines and iterators,
- exception handling,
- assertions, preconditions, postconditions, and class invariants.
- Sather programs can be compiled into portable C code and can efficiently link with C object files.



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Defining your own Classes

```
class MYPOINT is
   attr x, y:INT;
   create(xvalue, yvalue:INT):SAME is
      res ::= new;
      res.x := xvalue; res.y := yvalue;
      return res;
   end;
end;

X:MYPOINT := MYPOINT::create(3, 5);
Y ::= #MYPOINT(3, 5);
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Immutable Classes

```
immutable class MYPOINT is
    readonly attr x, y:INT;
    create(xvalue, yvalue:INT):SAME is
        res:SAME;
        res := res.x(xvalue);
        res := res.y(yvalue);
        return res;
    end;
end;

X:MYPOINT := MYPOINT::create(3, 5);
Y ::= #MYPOINT(3, 5);
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

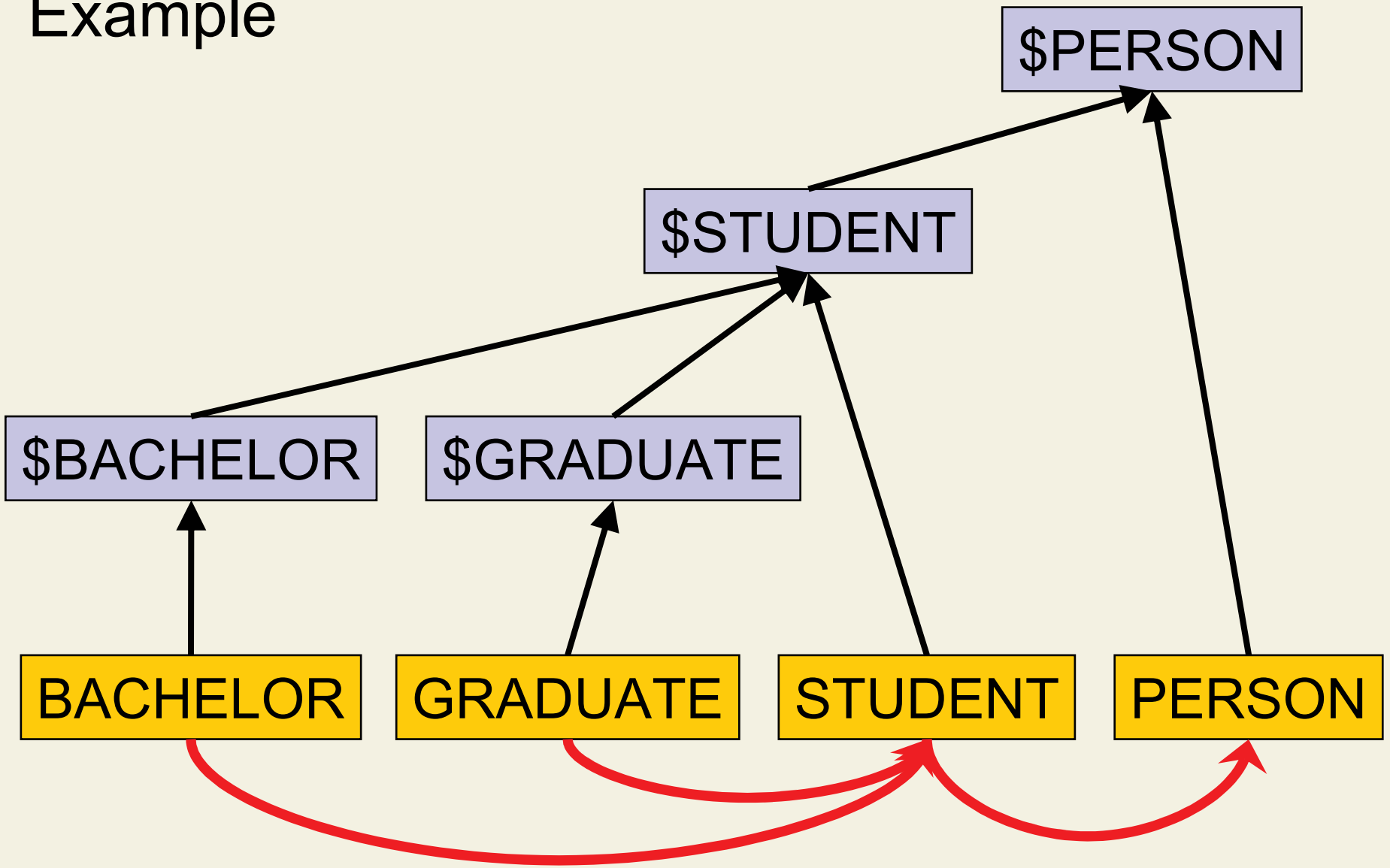
# Sather's Type System

- Sather distinguishes between type relations and code relations between classes
- Subtyping establishes a type relationship between classes, but no code is reused
- Multiple subtyping is allowed
- Even the concept of “supertyping” is introduced, which can be used to group common features of distinct type hierarchies

# Sather's Type System

- Code reuse is a separate concept, also called implementation inheritance or code inclusion
- Semantics of textual inclusion
- Inclusion from multiple classes allowed
- Features can be renamed and hidden
- Special partial classes exist that have no type and can not be instantiated. They can just be used for code inclusion into other classes

# Example






ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Abstract Classes

- Abstract classes only used for subtyping
- Not for code inclusion
- Class names have to start with a dollar $ and be in ALL CAPS
- Example:

```
abstract class $STACK is
        create:SAME;
        push(e:INT);
        pop:INT;
        is_empty:BOOL;
end;
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Subtyping

- A concrete or abstract class can be a subtype of an abstract class by using the < notation:

```
class ARR_STACK < $STACK is …
class LINK_STACK < $STACK is …
```

- Abstract classes can be used as attribute and method parameter types:

```
class USER is
    private attr stack:$STACK;
    create(s:$STACK):SAME is res ::= new;
        res.stack := s; return res;
    end;
end;
```

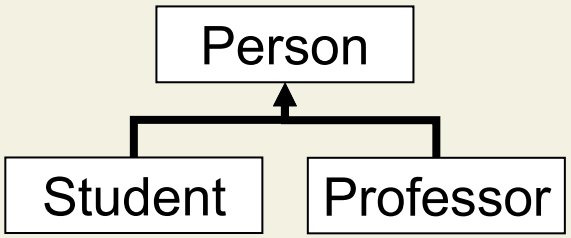


ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Supertyping

- In Sather it is possible to introduce types above other types by using >
- For example:

```
abstract class $IS_EMPTY > $LIST, $SET
is
    is_empty:BOOL;
end;
```




- This does not extend the interface of the subtypes, e.g. `$LIST` and `$SET` must already contain a feature `is_empty`

```
class READWRITE1 is
  attr mem:READWRITE2;

  getMem:READWRITE2 is return mem; end;

  create( a1:READWRITE2 ):SAME is
    res ::= new; res.mem := a1; return res;
  end;
end;

class READWRITE2 is
  attr mem1:CARD;

  create(a1:CARD):SAME is
    res ::= new; res.mem1 := a1; return res;
  end;
end;
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

```
abstract class $READONLY1 > READWRITE1 is
  mem:$READONLY2;

  getMem:$READONLY2;
end;

abstract class $READONLY2 > READWRITE2 is
  mem1:CARD;
end;

rw2 ::= #READWRITE2(4);
rw1 ::= #READWRITE1(2, 3.0, rw2);

ro1:$READONLY1 := rw1;
ro2:$READONLY2 := rw2;
ro3:$READONLY2 := ro1.getMem3;
```

# Inheritance without Subtyping

- Inheritance can be realized without subtyping
  - Example: Sather (a descendant of Eiffel)
- Using subclassing without establishing the “is-a” relation is problematic

```
class List {
  …
  void appendFront( Object o ) { … }
  void appendBack( Object o ) { … }
}
```

```
class Stack extends List {
  …
  // appendFront used as push
  void appendBack( Object o ) {
    System.out.println (“Should not
      be used!!”);
  }
}
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Inheritance without Subtyping

```
class List {
   …

 void appendFront(Object o)
   { … }

 void appendBack(Object o)
   { … }
}
```

```
class Stack
   extends List {

   hide List.appendBack;

   // appendFront is
   // used as push

   …
}
```

```
List l = someone.getList();

l.appendBack( new MyObject(…) );
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Sather Solution

```
class LIST is

 appendFront(p:$OB) is
    …
 end;

 appendBack(p:$OB) is
    …
 end;

end;
```

```
class STACK is

 include LIST
    appendBack ->;

end;
```

- Textual Code inclusion
- No subtype relationship
- → No problem with polymorphic calls



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Code Inclusion

- Include clause is used to include code from other classes
- Features can be renamed and modifiers changed
- A private include changes all modifiers to also be private
- Examples:

```
include A a->b, c->, d->private d;
private include D e->readonly f;
private include G h->h;
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Sather is Contravariant

- Assume the following classes:

```
abstract class $UPPER is …
abstract class $MIDDLE < $UPPER is …
abstract class $LOWER < $MIDDLE is …
```

- And a method:

```
abstract class $SUPER is
    foo( a1:$MIDDLE ): $MIDDLE;
end;
```

- Now what argument/return types can `foo` have in subtypes of `$SUPER`?



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Type Rules in Sather

- For routine signatures there are the following rules:
  - Arguments must have the same type or a **super**type
  - return values must have the same type or a **sub**type
- For the `$SUPER::foo` example from earlier:

```
class SUB < $SUPER is
    foo( a1:$UPPER ): $LOWER is …
end;
```

- Reminder of the substitution principle:
  *Objects of subtypes can be used wherever objects of supertypes are expected*



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Type Rules in Contracts

```
abstract class $SUPER is
      // requires type(a1) <= $MIDDLE
      // requires type(res) <= $MIDDLE
      foo( a1:$MIDDLE ): $MIDDLE;
  end;

class SUB < $SUPER is
      // requires type(a1) <= $UPPER
      // requires type(res) <= $LOWER
      foo( a1:$UPPER ): $LOWER is …
  end;
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# References

- Some links for Sather:
  - `http://www.icsi.berkeley.edu/~sather/`
  - `http://www.gnu.org/directory/devel/Programming_languages/sather.html`
  - `http://www.info.uni-karlsruhe.de/~sather/`
- Language descriptions and examples taken from: “A Language Manual for Sather 1.1” by Gomes, Stoutamire, Vaysman and Klawitter

# Questions?



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

# Hello World!

## Program in file Hello.sa:

```
class HELLO_WORLD is
   main is
         #OUT + "Hello World!\n";
   end;
end;
```

## Compilation:

```
sacomp -o Hello -main HELLO_WORLD Hello.sa
```



ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich