Selected Topics

Discussed Topics 2

Reactive programming

Concurrency in Eiffel (SCOOP), Smalltalk, Go

Micro-threads

Test driven development

Components in Java (EJB)

endless discussions what exactly a component should be ...

essential differences from objects, modules, classes:

compile-time unit, **delivered + required interfaces** specified,
deployment phase (connecting interfaces) between compilation and use

Enterprise Java Beans = component model

Java interfaces as delivered and required interfaces,
static or dynamic deployment,
separate name spaces $\longrightarrow$ serialization (frequently used option)

more recent: MicroService Architecture (MSA)

Templates in C++

Some Properties

constants usable as generic parameters:

```
template<typename T, int n> class buffer { ... };
```

(partial) specialization with “pattern matching”:

```
template<int n> class buffer<bool, n> { ... };
template<> class buffer<bool, 0> { ... };
```

implicit instantiation if no appropriate specialization available

code for methods produced only if used

Expressiveness

can be used to evaluate expressions statically (Turing-complete):

```
template<int x, int n> struct power {
    static const int r = x * power<x, n-1>::r;
};
template<int x> struct power<x, 0> {
    static const int r = 1;
};
```

algebraic data structures expressible:

```
template<class Head, class Tail> struct Cons { };
struct Nil { };
typedef Cons<int, Cons<float, Nil> > list_of_types;
```

Policy-Based Programming

```
#include <iostream>
template<typename lang> class HelloWorld : public lang {
   public: void Run() { cout << Message() << endl; }
};          // Message inherited from generic parameter

#include <string>
class HelloWorld_Msg_German {
   protected: string Message() { return "Hallo Welt!"; }
};

typedef HelloWorld<HelloWorld_Msg_German> MyHelloWorld;
MyHelloWorld hello;
hello.Run();
```

Alternative: Strategy Design Pattern (Java)

```
interface IMessage { String message(); }
class DMessage implements IMessage {
    public String message() {return "Hallo Welt!";}
}

class HelloWorld {
    private IMessage msg;
    public HelloWorld(IMessage msg) {this.msg = msg;}
    public void run() {System.out.println(msg.message());}
}

class Main {
    public static void main(String[] args) {
        HelloWorld hello = new HelloWorld(new DMessage());
        hello.run();
    }
}
```

Final Remarks

History of Object-oriented Programming

Languages: Simula, Smalltalk, Objective-C, C++, Eiffel, Self, CLOS, Oberon, Java, C#, Python, Ruby, ...

Concepts: structured programming, abstraction, inheritance, substitutability, interface specifications, parametrisation (genericity, annotations, aspects, ...)

Methods: factorization, use cases, graphical representation (UML), design patterns, pair programming, ...

Conflicts: functional programming, relational databases, collections and covariant problems, formal complexity, concurrency

Trends: object-based, object-oriented, partially automated, typed, team+architecture-integrated, layers and frameworks, back to the roots

Future of Object-oriented Programming

OOP omnipresent $\longrightarrow$ no longer innovativ

splitting up into many details and side issues

topics of the near future: concurrency, distributed programming, data integration and big data, cloud computing, complex behavioural interfaces, security, ...

currently more open questions than answers

language support expected when most important questions answered

$\longrightarrow$ language support mainly for topics that are no longer up-to-date?