Selected Topics

Visibility and Security

visibility on class level (Java, C#, C++, ...)

→ focus on **responsibilities** of class

visibility on object level (variables in Eiffel)

→ focus on **substitutability** and assertions

visibility as a **security concept** (not in mainstream languages)

→ constraints on references to objects:

uniqueness: only one reference to object can exist
ownership: only referred to by one object
confinement: only referred to within a specific region

Escape Analysis

compiler (including JIT) analyses object use

if object stays in local scope (non-escape), then

- confined to scope
- object variables on stack or in other object
- cheap allocation, further optimizations possible

else if object stays within thread, then

- confined to thread
- no synchronization necessary

used e.g. in Java HotSpot VM for optimization
and in language extensions / tools to give security guarantees

Java Modeling Language (JML)

supports formal DbC assertions as Java comments

different kinds of tools use these assertions

> Java compiler adding run-time checking as in Eiffel,
> static checking (with help by programmers),
> testing tool in combination with JUnit,
> documentation generator

rather conventional assertions expressible

> preconditions, postconditions, invarinats,
> local assertions and visibility, model variables
> quantified variables

no history constraints, concurrency and substitutability, ...

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,
deployment statically or dynamically,
separate name spaces $\longrightarrow$ serialization (frequently used option)

Serialization in Java

if interface `Serializable` is implemented, serialization can occur automatically by `ObjectOutputStream.writeObject(Object obj)`,
deserialization by `ObjectInputStream.readObject()`

Variables declared as `static` or `transient` are ignored

for special care (e.g., `transient`) these methods must be implemented:

```
private void writeObject(java.io.ObjectOutputStream out)
    throws IOException;
private void readObject(java.io.ObjectInputStream in)
    throws IOException, ClassNotFoundException;
private void readObjectNoData()
    throws ObjectStreamException;
```

to avoid conflicts with wrong versions of classes (should change with versions):

```
... static final long serialVersionUID = ...;
```

Some Notions

aspect-oriented programming: separation of concerns

subject-oriented programming: object differs from implementation

mixin: often synonym for trait or similar things, not exactly defined

TypeScript: preprocessor language for JavaScript (Microsoft)

Functional Java: API for functional programming in Java

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 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?