# TTCN-3 edition 2017/18

# – Object-Orientation and More –

## TTCN-3 at a glance

- ***THE*** global testing language
- First version published in 2000, i.e. TTCN-3 is based on 17 years of practical experience, continuous maintenance, and further development
- **Supported types of testing:** conformance/functional, load, performance, interoperability, and security testing
- ***TTCN-3 application areas:*** telecommunications, automotive, transportation, Internet, medical, web-based services, finance, industrial automation, distributed systems, and many more
- Further information and possibilities for contributing to TTCN-3 can be found on: http://www.ttcn-3.org/

Intelligent Transport Systems © ETSI

## The TTCN-3 Series of Standards

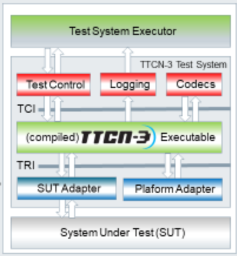
**TTCN-3 core testing concepts**
- ES 201 873-1 "Core Language"

**Using TTCN-3 with other languages**
- ES 201 873-7 "The use of ASN.1"
- ES 201 873-8 "The IDL to TTCN-3 Mapping"
- ES 201 873-9 "Using XML schema with TTCN-3"
- ES 201 873-11 "TTCN-3: Using JSON with TTCN-3"

**TTCN-3 documentation support**
- ES 201 873-10 "Documentation Comment Specification"

**TTCN-3 beyond functional conformance testing**
- ES 202 781 "Configuration & Deployment Support"
- ES 202 782 "Performance and Real Time Testing"
- ES 202 784 "Advanced Parameterization"
- ES 202 785 "Behaviour Types"
- ES 202 786 "Support of Interfaces with Continuous Signals"
- ES 202 789 "Extended TRI"
- ES 203 022 "Advanced Matching" (NEW 2017)
- ES 203 790 "Object Oriented Features" (NEW 2018)

**Implementation of TTCN-3 test systems**
- ES 201 873-4 "Operational Semantics"
- ES 201 873-5 "TTCN-3 Runtime Interface (TRI)"
- ES 201 873-6 "TTCN-3 Control Interface (TCI)"



TTCN-3 test system © ETSI

## Evolving TTCN-3 towards Object-Orientation

- Heighten appeal of TTCN-3 to users used to object-oriented programming
- Use advantages of object-oriented modelling
- Reduce TTCN-3 emulation of object-oriented features
- Allow simple access to external objects

## Object-Oriented Concepts: Classes

**Modelling of objects**
- Member fields (state)
- Virtual methods (behavior)

**Built-in classes**
- `object`, `component`, `timer`, `port`

```
type class Vehicle {
  var integer velocity := 0;
  public function accelerate(integer v) {
    velocity := velocity + v;
  }
  public function getVelocity() return integer
{ return velocity; }
}
type class Car extends Vehicle {
  public const charstring license_plate;
}
```

Class definition and class refinement

**Allow abstraction, refinement and encapsulation**
- Class extension provides subtypes
- Method overriding
- Member visibility
  - `public`, `private`, `protected`

**Provide API to external objects**
- External classes

```
external type class Stack {
  function pop();
  function push(integer x);
  function top() return integer;
}
```

Definition of an external class

## Object-Oriented Concepts: Objects

- Used by reference
- Belong to creating component context
- Provide handles to external objects
- Class discrimination (`o of C`)
- Casting (`C:o`)
- Direct method/field access from inside owning component context (`o.f`)
- Special references '`null`' and '`this`'.
- Object-identity equality comparisons ( `==`, `!=` )
- Created via constructor (`C.create(...)`)
- Implicit memory management

## Exception Handling

**Cleaner test code**
- separated exception handling

**Cleaning up & resetting external resources**
- as part of the standard
- "`finally`"

**Can be validated statically**

**Reusing existing keywords**
- `exception`, `raise`, `try`, `catch`

```
testcase t_myTest() runs on CT {
   f_init("user1");
   f_init("unknown");   // bad argument
   :
} finally {
   :          // resources may be freed
              // dynamic error is reported
}
```

Example for the use of `finally`

```
function f_init(in charstring name) exception (charstring) {
   :
   if (name_was_not_registered) {
    raise charstring:("Could not init " & name);
   }
   :
}
function f_reg(in charstring user1, in charstring user2) {
   f_init(user1);
   f_init(user2);
   :
} catch (charstring e) {
   : // exception is available for processing in the e variable
}
```

Defining exception handling

## TTCN-3 Extension: Advanced Matching

New and powerful matching mechanisms for TTCN-3:

- **Dynamic matching**
  Define your matching in form of a function!
- **Templates with variable bindings**
  Field values in case of successful matching can be specified as out-parameters of template definitions

```
external function fx_isPrime
         (integer p_x) return boolean;
:
p.receive(@dynamic fx_isPrime)
// is the same as
p.receive(integer: @dynamic {
         return fx_isPrime(value)})
```

Example for dynamic matching

- **Logical operators for combining matching mechanisms**
  **New operators:** conjunction, implication, exclusion, and disjunction
- **Repetition**
  Support to match repetitions of sub-sequence templates inside values
- **Restrictions for omit symbol and templates with omit restriction are relieved**
  Omit symbols and templates with omit restriction may be used as operands for the equality operator

5th UCAAT User Conference on Advanced Automated Testing

Berlin, 11-13 October 2017

ETSI STF 533 on "TTCN-3 Evolution 2017"

J. Grabowski, J. Deltour, K. Szabados, G. Réthy, T. Urban, J. Wieland

stf533@etsi.org