The object-oriented computational paradigm

Computational paradigm =
high-level programming languages sharing the same execution model

Main computational paradigms

Paradigm	**Model**	**Launching a program corresponds to**
Imperative	state as memory abstraction	execute a command
Object-oriented	object world and message sending	send a message to an object
Functional	functions and application	evaluate an expression
Logic	Horn clauses and deduction	resolve a goal

Remarks: most modern programming languages support several paradigms!

A brief history

First object-oriented languages

- Simula (1965)
- Smalltalk (1972)
- Eiffel (1986)

Mainstream object-oriented languages

- statically typed: C++ (1985), Java (1995), C# (2000)
- dynamically typed: Python (1990), JavaScript/ECMAScript (1995)

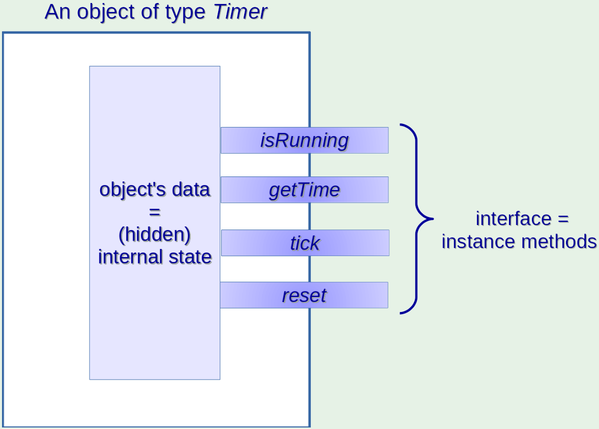
Most recent ones

- Scala (2004), strong integration of functional and o-o paradigms
- Kotlin (2011), Java “enhancement”, support for Android

Both statically typed and based on the Java Virtual Machine (JVM)

What is an object?

Example: a timer



Instance methods

How can we interact with an object?

- It is possible to call the instance methods of the object
- Equivalent terminology: method invocation or message sending
- Terminology: the object is the target or receiver of the instance method
- The execution of a instance method may
  - modify the data (=internal state) of the object
  - pass some arguments
  - return a value

Syntax of method call

```
Exp ::= Exp '.' MID '(' (Exp ( ',' Exp)*)? ')'
```

- `MID`: instance method identifier (= name)
- Example:

```
// instance method reset called on target timer with argument 42
timer.reset(42);
```

Instance methods

Specification of timer instance methods

- `boolean isRunning()`
  - checks whether the count down is not finished yet
  - returns true if and only if the timer has not reached time 0
- `int getTime()`
  - returns the current time of the timer expressed in seconds
- `void tick()`
  - decreases of one second the time of the timer if it is greater than 0
  - no action is taken if the time is 0
- `int reset(int minutes)`
  - resets to `minutes` the time of the timer
  - `minutes` is expressed in minutes
  - returns the previous time (in seconds) of the timer
  - throws an exception if `minutes<0` or `minutes>60`

Instance methods

A closer look at the instance methods of a timer object

- query methods: `isRunning` and `getTime`
  - they inspect the data (=internal state) of the timer object
  - they cannot modify them
- `tick` may modify the data (=internal state)
- `reset` inspects the data (=internal state) and may modify them

Object's data = internal state

General facts

- usually hidden to the external world
- consists of fields
- fields are also called instance variables or attributes
- instance variables store the data (=values) of an object
- fields can usually be updated

Two possible implementations of the internal state of a timer

-
  ```
  // total time expressed in seconds
  int time; // invariant: 0 <= time <= 3600
  ```

-
  ```
  // total time is seconds+60*minutes
  int seconds; // invariant: 0 <= seconds < 60
  int minutes; // invariant: 0 <= minutes <= 60
  ```