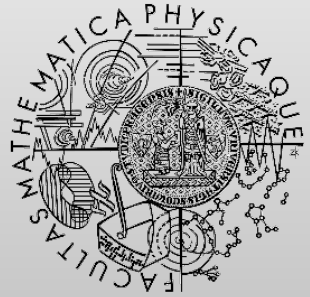
Statically-typed Class-based languages – Scala

http://d3s.mff.cuni.cz



FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

Tomas Bures

bures@d3s.mff.cuni.cz

Scala

- Statically-typed language
- Compiles to bytecode
- Modern concepts

- Example: E01

Syntax inference

- A line ending is treated as a semicolon unless one of the following conditions is true:
  - The line in question ends in a word that would not be legal as the end of a statement, such as a period or an infix operator.
  - The next line begins with a word that cannot start a statement.
  - The line ends while inside parentheses (...) or brackets [...], because these cannot contain multiple statements anyway.

- Blocks are based on indentation
  - Possible to use curly braces (version 2 syntax)

Static vs. dynamic typing

- Target function is determined
  - at compile time – static typing
  - at runtime – dynamic typing

- Example: E02

Classes vs. objects

- Scala does not have static method
- Instead it features a singleton object
  - Defines a class and a singleton instance

- Example: E03

- Decompiled – AppLogger, Logger

Type inference

- Types can be omitted – they are inferred automatically
  - At compile time

- Example: E04

Type Hierarchy

- Everything is an object
  - primitive data types behind the scene (boxing/unboxing)
- Compiler optimizes the use of primitive types
  - a primitive type is used if possible

Companion object

- A class and object may have the same name
  - Must be defined in the same source
- Then the class and object may access each others private fields
- Example: E05

Constructors

- One primary constructor
  - class parameters
  - can invoke superclass constructor
- Auxiliary constructors
  - must invoke the primary constructor (as the first one)
  - must not invoke superclass constructor

Namespaces

- Scala allows groups classes to packages (similar to Java and C#)
- Similar to C#, it allows defining multiple classes and even packages in the same file

- Example: E06

Operators

- Scala allows almost arbitrary method names (including operators)
- A method may be called without a dot
- Prefix operators have special names

- Example: E07

Flexibility in Identifiers and Operators

- Alphanumeric identifier
  - starts with letter or underscore
- Operator identifier
  - an operator character belongs to the Unicode set of mathematical symbols(Sm) or other symbols(So), or to the 7-bit ASCII characters that are not letters, digits
  - any sequence of them
- Mixed identifier
  - e.g. unary_- to denote a prefix operator
- Literal identifier
  - with backticks (e.g. `class`) to avoid clashes with reserved words, etc.

Operator precedences

- Operator precedence determined by the first character
  - Only if the operator ends with “=“, the last character is used

(all other special characters)
* / %
+ -
:
= !
< >
&
^
|
(all letters)
(all assignment operators)

Extensions

- Similar to C#, Scala makes it possible to declare an extension of an existing type
- The extensions have to be brought to scope
  - Typically imported
- Example: E08

Context parameters (aka givens)

- Scala allows naming instances (called “givens”) that define canonical values of certain types
  - used to synthetize arguments for context parameters
- Givens have to be brought to scope to be applicable
  - Special import notation
- Example: E09

Implicit conversions

- Scala allows specifying functions that are applied automatically to make the code correct
  - conversion to the type of the argument or to the type of the receiver
  - the conversion is brough in as a “given” – same rules apply for making it visible as for other givens

- Example: E10 + H1

Rich wrappers

- Implicit conversions used to implement so called Rich wrappers
- Standard library contains rich types for the basic ones
  - E.g. RichInt – defines methods to, until, …

First-class functions

- Functions are first-class citizens
- May be passed as parameters
- Anonymous functions, …
- Anonymous functions are instances of classes
  - Function1, Function2, …

- Example: E11

Tail recursion

- The compiler can do simple tail recursion
  - If the return value of a function is a recursive call to the function itself
- Example: E12

For-comprehension

- Generalized for-loops
  - generators, definitions, filters
- Translated to operations over collections
  - map, flatMap, withFilter, foreach
- Example: E13