

The DEV team

INFDEV02-3

January 11, 2016

1 Lectures and homework

1.1 Week 1 - statically typed programming languages

Topics

- What are types?
- (**Advanced**) Typing and semantic rules: how do we read them?
- Introduction to Java and C# (**advanced**) with type rules and semantics
 - Classes
 - Fields/attributes
 - Constructor(s), methods, and static methods
 - Statements, expressions, and primitive types
 - (**Advanced**) Lambda's

Homework

- Write an example of Python code that would cause a type error in Java/C#
- Given the following semantic and typing rules, write down how we read them; make an example code that uses them
- Write a Java/C# program featuring
 - A `Counter` class;
 - With a `count` integer attribute;

- With an empty (parameterless) constructor;
- With a method `Reset`;
- With a method `Tick`;
- (**Advanced**) With a static method/overloaded operator `Plus` which adds two counters into one;
- (**Advanced**) With a method `OnTarget` that takes as input a lambda function which will be fired when the counter reaches a given count.

1.2 Week 2 - reuse through polymorphism and generics

Topics

- What is code reuse?
- Interfaces and implementation
- Implicit vs explicit conversion
- (**Advanced**) Implicit and explicit conversion type rules
- Runtime type testing
- (**Advanced**) Generic parameters
- (**Advanced**) Interfaces and implementation in the presence of generic parameters
- (**Advanced**) Covariance and contravariance in the presence of generic parameters

Homework

- Write a `Vehicle` interface with a method `move` and a method `loadFuel`; `loadFuel` accepts a `Fuel` instance, where `Fuel` is an interface of your writing; `move` returns a boolean which is `true` if there is enough fuel, and `false` otherwise
- Write a concrete class `Car` and a concrete class `Gasoline` that implement, respectively, `Vehicle` and `Fuel`; the `Car` checks that the given fuel is indeed `Gasoline`

- Write a concrete class **Truck** and a concrete class **Diesel** that implement, respectively, **Vehicle** and **Fuel**; the **Truck** checks that the given fuel is indeed **Diesel**
- Write a concrete class **Enterprise** and a concrete class **Dilithium** that implement, respectively, **Vehicle** and **Fuel**; the **Enterprise** checks that the given fuel is indeed **Dilithium**
- Make a program that receives three vehicles, without knowing their concrete type, and moves them (without resorting to conversions) until their fuel is up
- (**Advanced**) Make a **List<T>** interface with methods **Length**, **Iterate**, **Map**, and **Filter**
- (**Advanced**) Define the concrete classes **Node<T>** and **Empty<T>** both implementing **List<T>**
- (**Advanced**) Make a **List<Vehicle>**, fill it with a series of concrete vehicles, and make them all move ten times