

DCGI

DEPARTMENT OF COMPUTER GRAPHICS AND INTERACTION

WA2

Cvičení 6

Úvod do .NET, ASP.NET, MVC3

Test

- TODO: 10+ min



.NET Framework objectives

- provide a consistent OO programming environment
- minimise software deployment and versioning conflicts
- promote safe execution of code, including code created by an unknown or semi-trusted third party
- eliminate the performance problems of scripted or interpreted environments
- make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications
- ensure that code based on the .NET Framework can integrate with any other code

.NET Framework

1.0	2002	initial version
1.1	2003	ASP.NET, ODBC, Compact FW, API changes
2.0	2005	Generics, 64bit, SQL Server integration, Micro FW, Partial Classes, API changes
3.0	2006	WPF (Presentation), WCF (Communication), WF (Workflow), Windows CardSpace
3.5	2007	LINQ, ADO.NET Entity FW, ADO.NET Data Services
4.0	2010	Parallel Extensions, PLINQ
4.5	2012	Metro Style Apps, various core improvements, MEF

.NET Languages

- **C#, J#, VB.NET, C++/CLI**
- A#, Boo, Cobra, Component Pascal, F#, IronPython, IronRuby, IronLisp, JScript .NET, L#, Managed JScript, Nemerle, Oxygene, P#, Phalanger, Phrogram, Windows PowerShell

http://en.wikipedia.org/wiki/List_of_CLI_languages

.NET Assembly

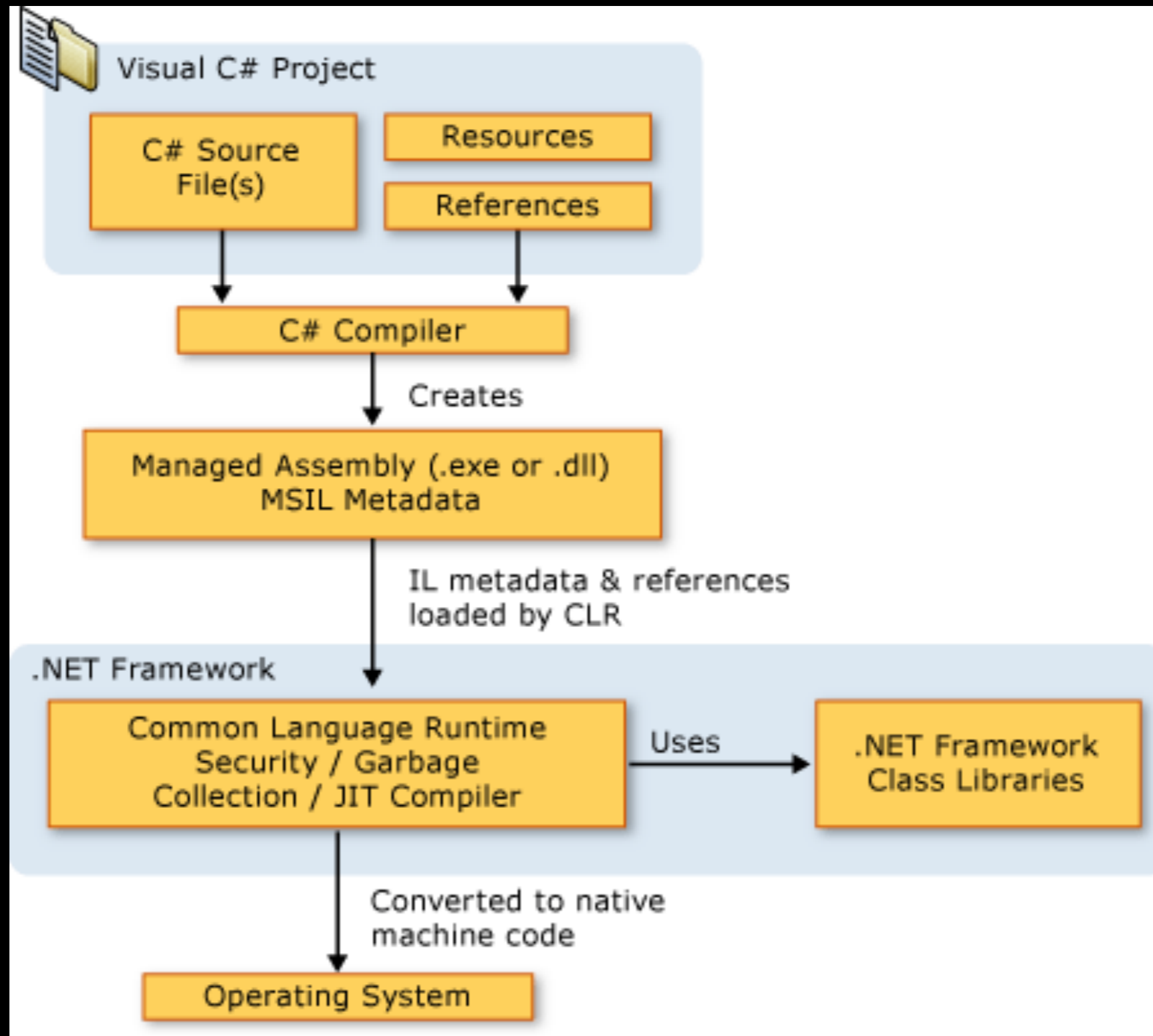
- MSIL
- PE file (an .exe or .dll)
- Manifest
- Entry Point
- Boundary (security, type, version)
- Deployment unit
- Static or dynamic.

Assembly manifest

- Every assembly (static or dynamic)
- Contains assembly metadata
 - Assembly name
 - Version number
 - Culture
 - Strong name information
 - List of all files in the assembly
 - Type reference information

.NET Framework

C#

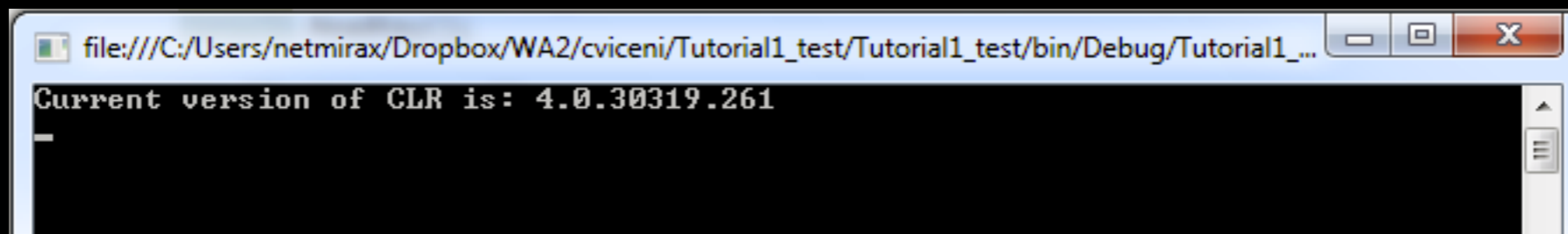


C# .NET

- “main” .NET language
- simple, modern, general-purpose, object-oriented programming language
- mainly influenced by C++ and Java (and Eiffel, Modula-3, Object Pascal)
- Garbage collection
- strong type checking, array bounds checking

Task 1

1. Start Visual Studio
2. Create new project (and solution): File → New → Project → Visual C# → Console Application
3. Use .NET Framework Class Library to display current version of CLR. Output: console. (hint: Environment)



A screenshot of a console application window. The title bar shows the file path: file:///C:/Users/netmirax/Dropbox/WA2/cviceni/Tutorial1_test/Tutorial1_test/bin/Debug/Tutorial1_... The main content area displays the text: Current version of CLR is: 4.0.30319.261

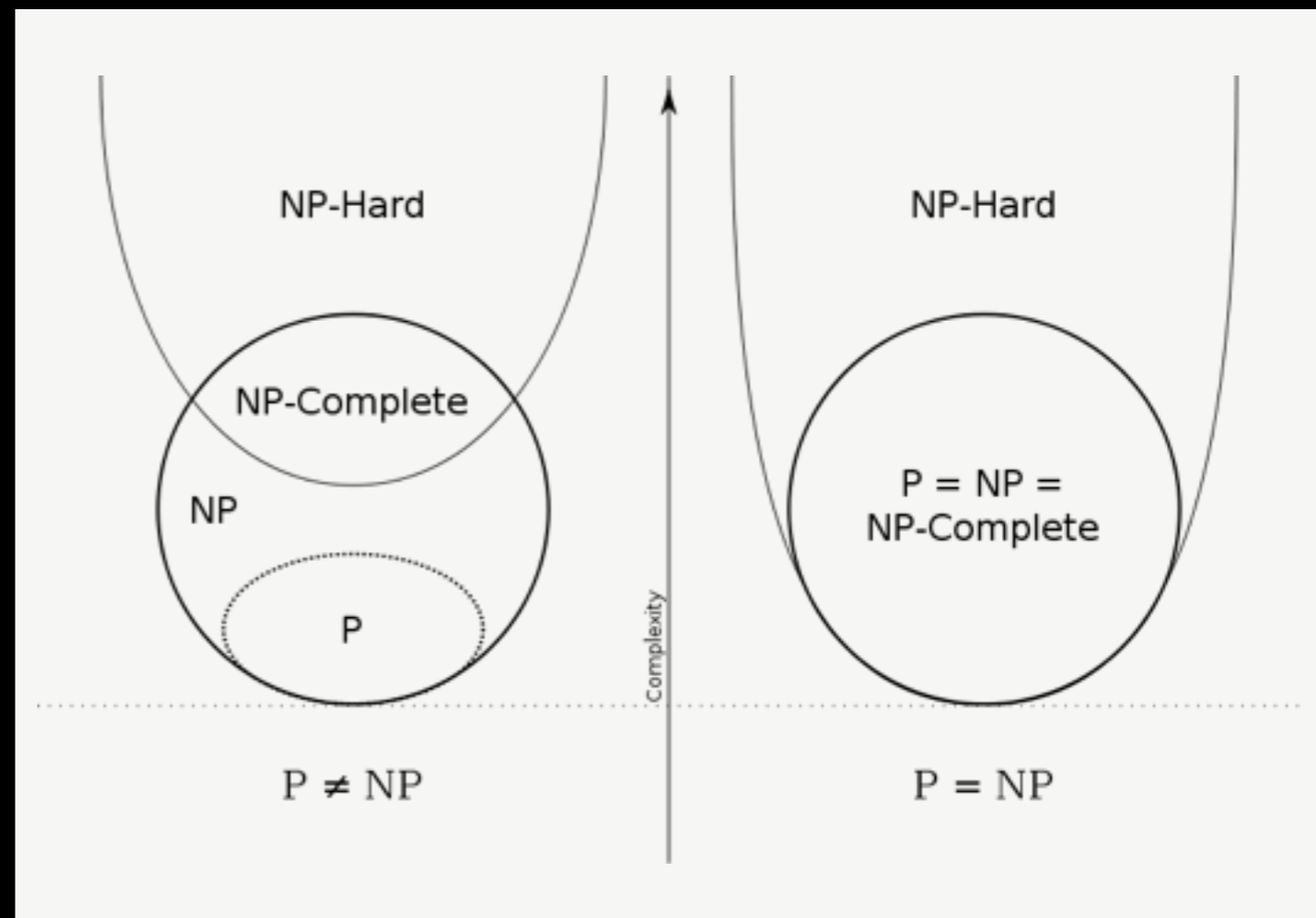
Task 2

1. Add reference to provided class library (dll assembly)
2. Study library contents
3. Using inheritance extend TSP.Algorithm.City implementation in order to store city names.
4. Use library for TSP (Travelling Salesman Problem) computation. Discussion: TSP, NP, NP-complete, Simulated evolution.
5. Report results to console

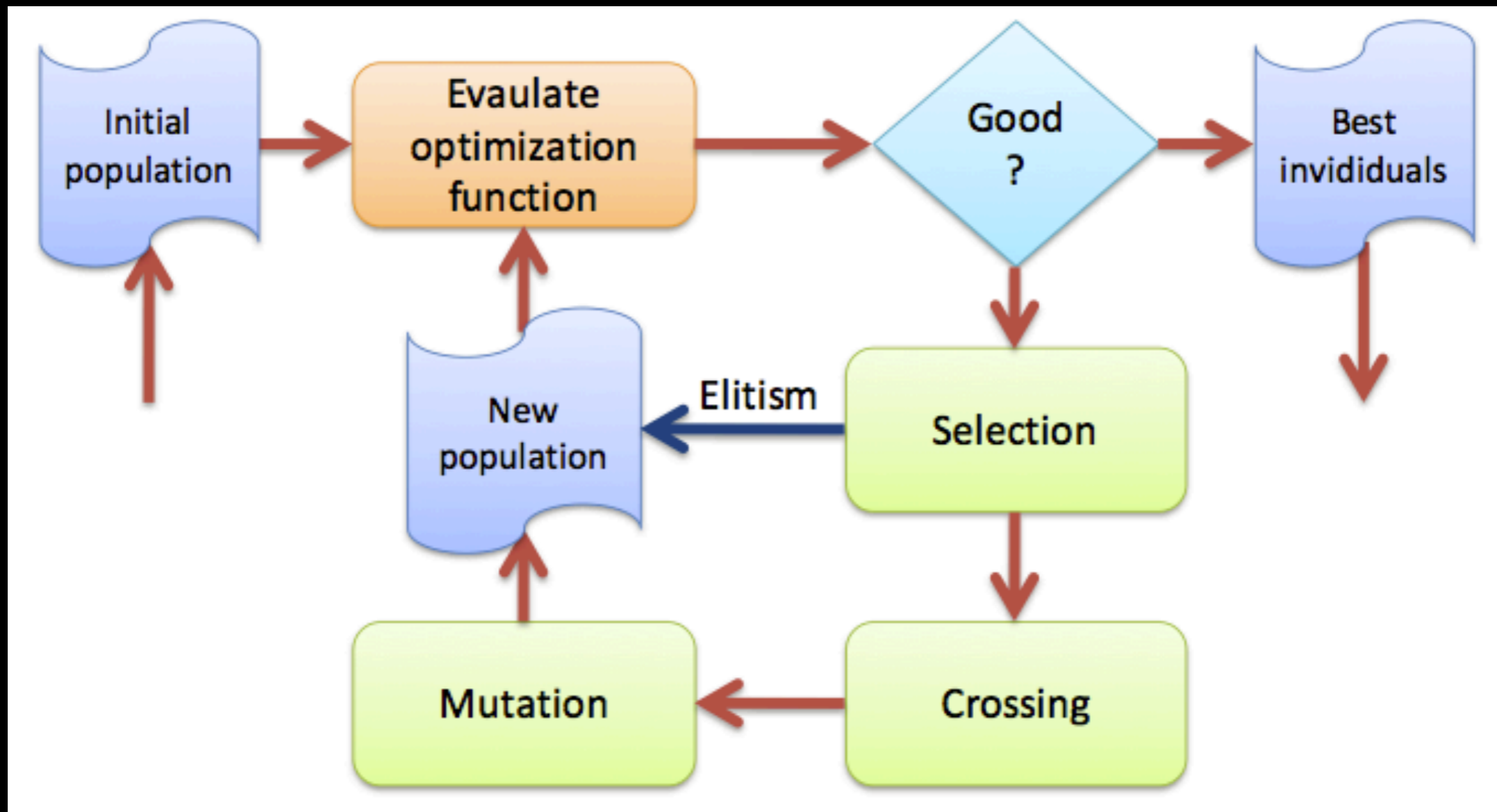
```
Best cost so far: 29505.9035875497, generation 80
Best cost so far: 29505.9035875497, generation 81
Best cost so far: 29505.9035875497, generation 82
Best cost so far: 29505.9035875497, generation 83
Best cost so far: 29505.9035875497, generation 84
Cost of minimal route found: 29505.9035875497, generations: 85
Rome -> Innsbruck -> Prague -> Berlin -> Paris -> London -> Newcastle upon Tyne -
> NYC -> Las Vegas -> L.A. -> San Francisco -> Honolulu -> Seoul -> Fukuoka -> K
ota Kinabalu -> Kuala Lumpur ->
```

TSP

- NP-Hard
- Non deterministic Polynomial time - Hard
- NP-Complete
- Non deterministic Polynomial time - Complete



Simulated Evolution Genetic Algorithm

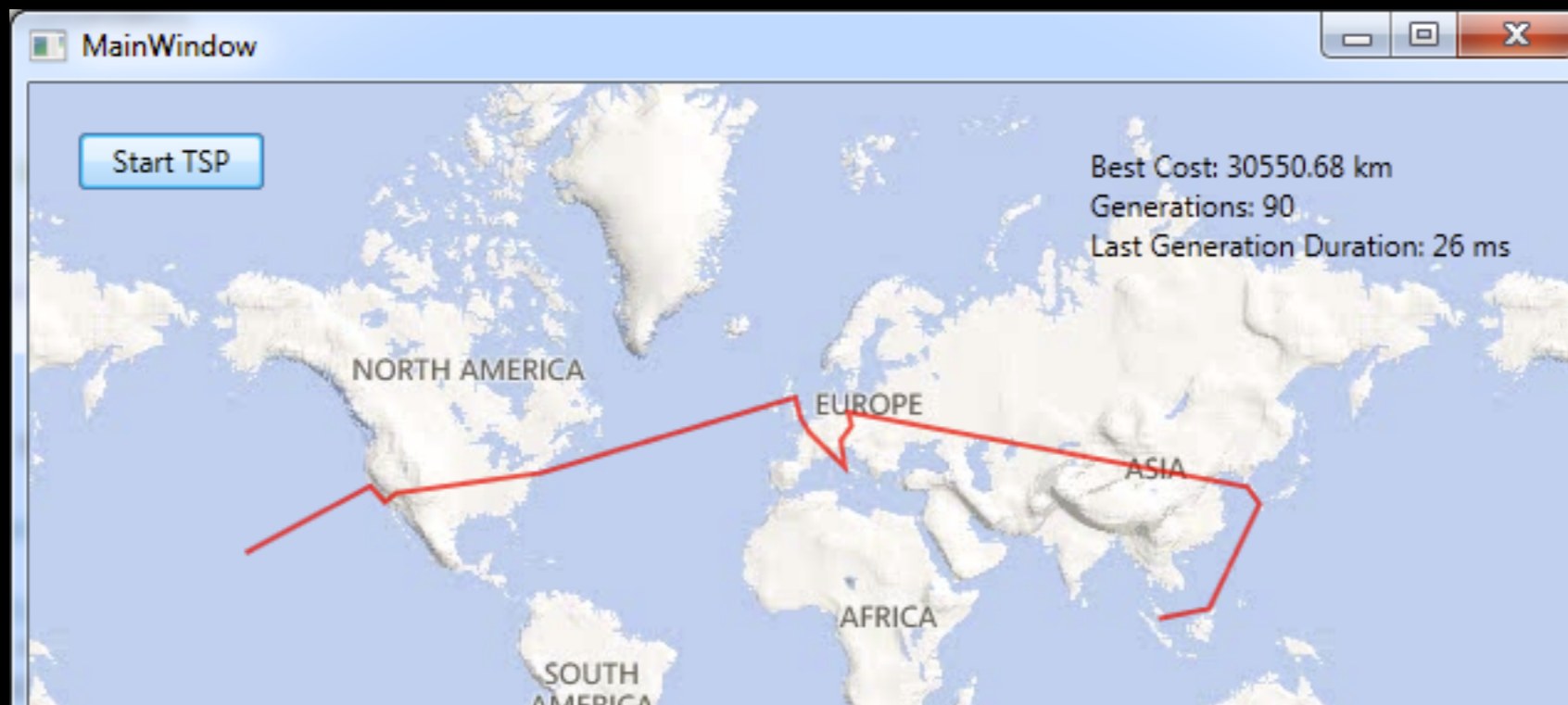


Task 3

1. Add new project (name TSPSolverGA). Type: Class library
2. Reference to provided class library
3. Implement provided interface ITSPSolver
 - TSP computation should run in a separate thread (use code fragments below)
4. Implement provided interface ICity interface as City class, mind namespaces.
 - Either extend (inherit) implementation of TSP.Algorithm.City.
 - Or implement the ICity interface from scratch

Task 4

1. Add project WpfTSP to your solution
Reference TSPSolverGA project from previous task.
2. Reference SolverInterface
3. Complete unfinished methods, use implementation from previous task.



Task 5

1. Add reference to provided class library (dll assembly) - TSP Algorithm to WPF Project.
2. Use last parameter of ExecutionFinished respective ProgressReached events and check whether it is GeneticAlgorithm. If true display in info label following parameters:
 - Best cost
 - Count of generations
 - Last generation duration

Task 6

- Perform performance analysis of your solution (Debug → Start Performance Analysis)

Home work

- Implement a TSP solver that always finds optimal solution. Use `ITSPSolverInterface`. You can either use naive method or for better solution for extra points (e.g. branches and bound).