Comparaison machines virtuelles Java/C#

Ludovic Feltz
ESIPE MLV

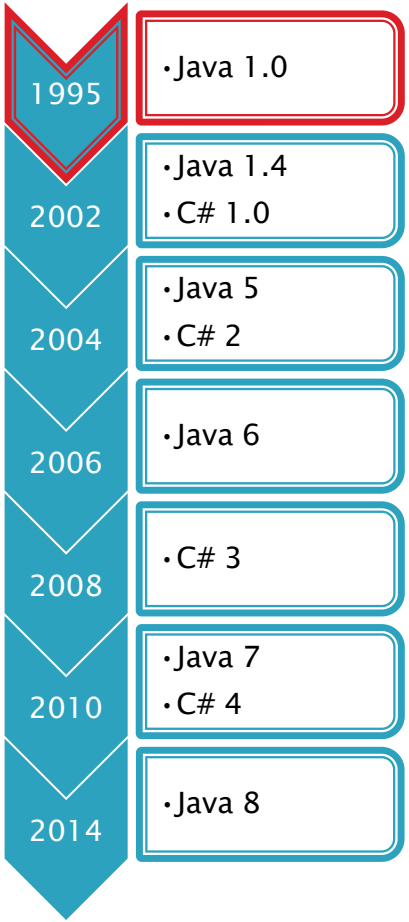
Introduction

- Historique et présentation des similarités et des différences entre les VM.

Plan

- Historique des langages
- Les VMs
- Les différences entre les VMs
- Futures évolutions
- Démonstration
- Conclusion

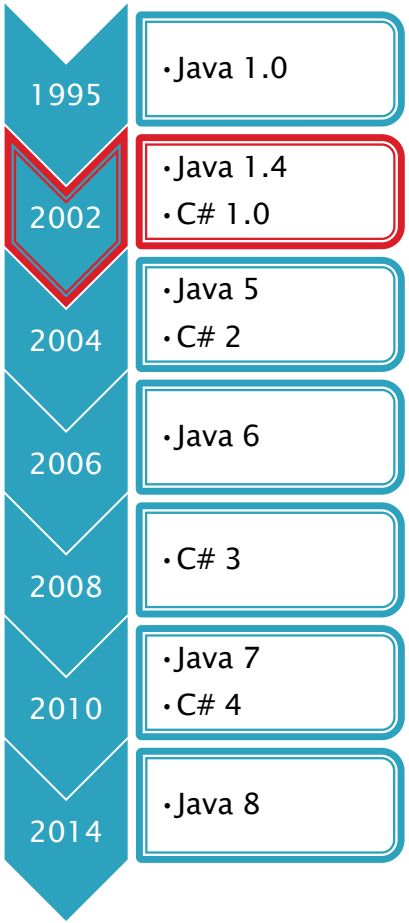
Historique



1995:

- Sortie de Java développé par Sun
  - Langage orienté objet
  - Exécutable sur tous les systèmes.
  - Applets Java rapidement adoptés par les navigateurs.

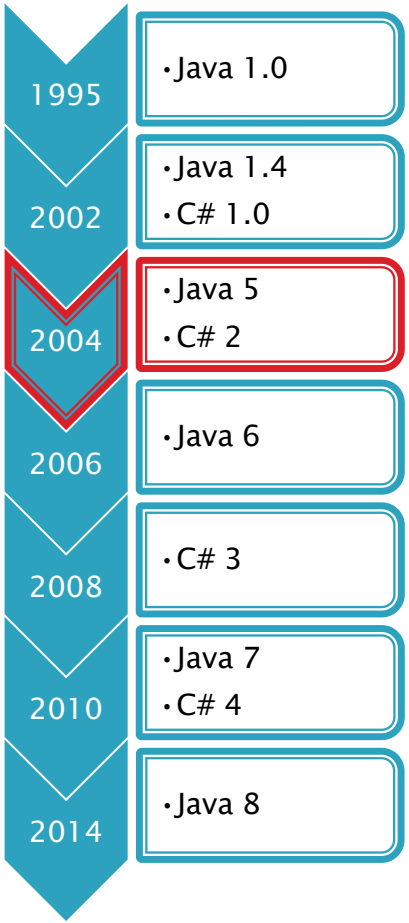
Historique



2002:

- Java 1.4:
  - JIT (just in time)
  - Optimisation (hotspot)
- Sortie du langage C#
  - Appelé imitation du java dont il est inspiré.
  - Sa syntaxe plus proche du C++.

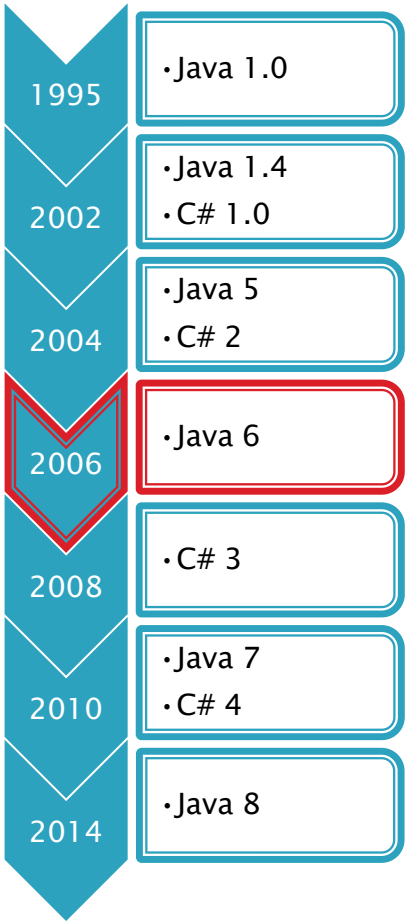
Historique



2004:

- Java:
  - Types génériques
  - AutoUnboxing
  - Annotation
  - Enumeration
  - Varargs
- C#:
  - Types génériques
  - Delegates

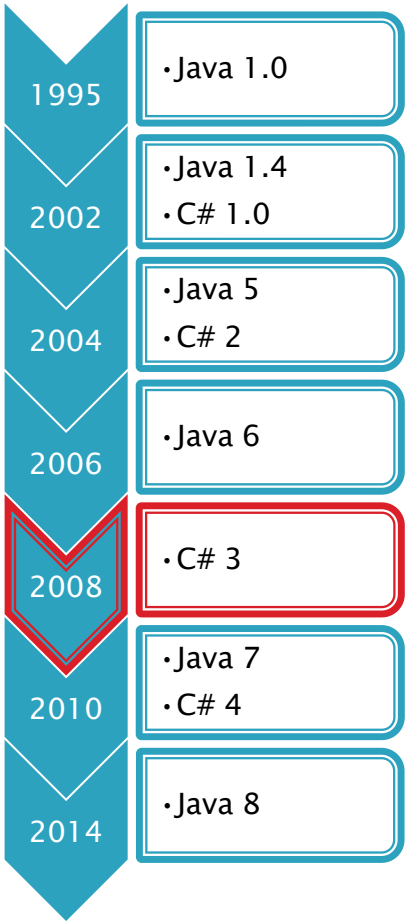
Historique



2006

- Java:
  - optimisations hotspot

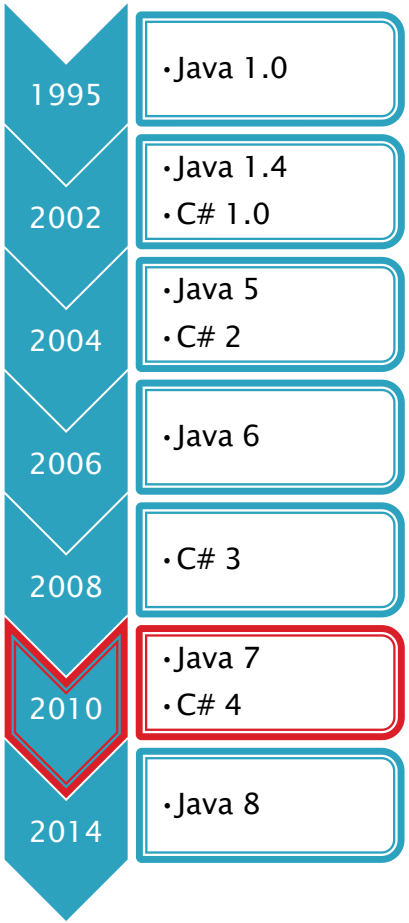
Historique



2008:

- C#:
  - typage implicite
  - Lambda
  - Linq

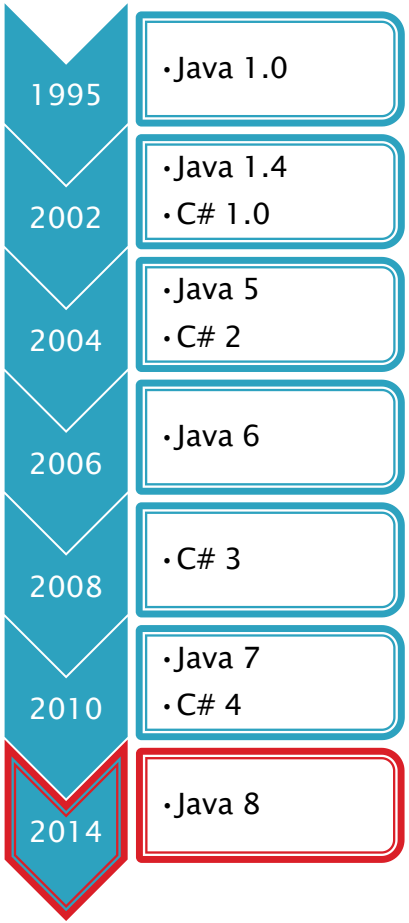
Historique



2010:

- Java:
  - Operateur Diamond
  - String dans les switchs
- C#:
  - Typage dynamique
  - Arguments nommés

Historique



2014:

- Java:
  - Lambda
  - Amélioration Date et Time API

Qu'est ce qu'une VM ?

- VM (Virtual Machine): Simule l'architecture d'une machine.
- Les VM servent à exécuter du code managé (code dans un langage intermédiaire).
- Une même application peut être exécutée sur chaque système possédant une version de la VM.

Qu'est ce qu'une VM ? (2)

- C'est un software au dessus du harware.
- Pas d'appel aux registres du processeur ou d'utilisation de la mémoire.
- Elle s'occupe de gérer les objets en mémoire: allocation et destruction (garbage collector)
- Gestion des threads

Les VMs

Selon le langage les VM sont différentes:

- **Java**: on utilisera une implémentation de la **JVM** dépendant du système.

- **C#**: **CLR** pour Windows ou **Mono** pour Linux, Mac, …

Java et la JVM

- L'implémentation d'une JVM n'est pas définie.
  - Doit être conforme aux spécifications de Sun

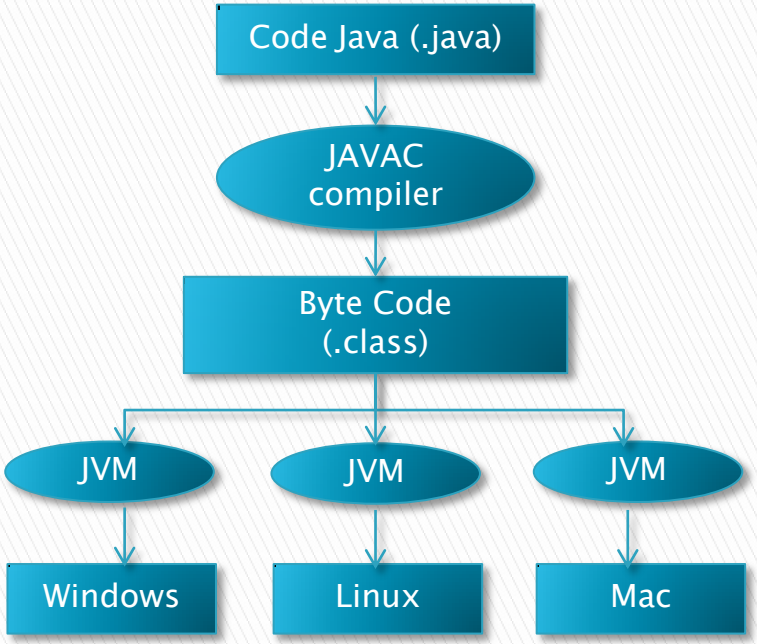
- Elle exécute un byte code. (fichiers .class)
  - Set d'instructions

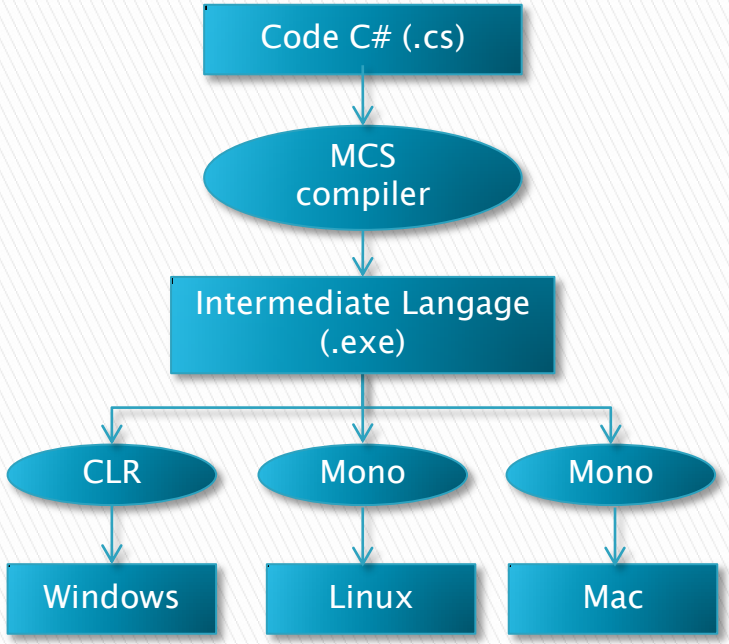
C# et CLR/Mono

- Deux VM existent pour le langage C#:
  - **CLR**: développé par Microsoft pour un système Windows.
  - **Mono**: VM libre permettant d'exécuter du C# sur d'autres plateformes.

- Elles exécutent un code intermédiaire: MSIL (Microsoft Intermediate Language).

Compilation/Execution d'un programme



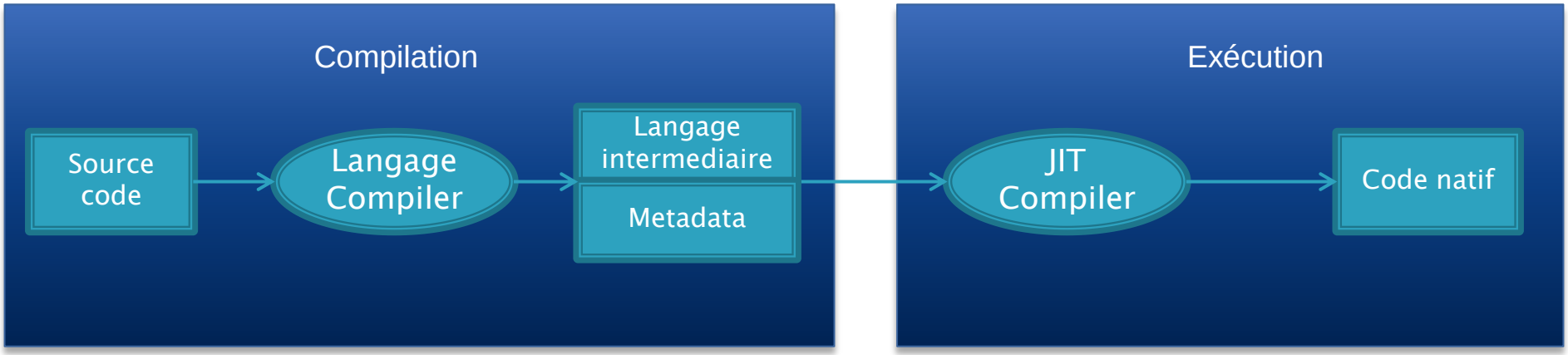


Java

C#

Execution JIT (Just In Time)

- Les VM utilisent la compilation JIT
  - Le code est interprété et compilé en langage machine pendant son exécution



MSIL (C#) vs ByteCode (Java)

```
.class public auto ansi beforefieldinit BasicProgram.BasicProgram
    extends [mscorlib]System.Object
{
    // Fields
    .field private int32 'value'

    // Methods
    .method private hidebysig
        instance void set (
            int32 'value'
        ) cil managed
    {
        // Method begins at RVA 0x2050
        // Code size 9 (0x9)
        .maxstack 8

        IL_0000: nop
        IL_0001: ldarg.0
        IL_0002: ldarg.1
        IL_0003: stfld int32 BasicProgram.BasicProgram::'value'
        IL_0008: ret
    } // end of method BasicProgram::set

    .method public hidebysig static
        void Main (
            string[] args
        ) cil managed
    {
        // Method begins at RVA 0x205c
        // Code size 17 (0x11)
        .maxstack 2
        .entrypoint
        .locals init (
            [0] class BasicProgram.BasicProgram bp
        )

        IL_0000: nop
        IL_0001: newobj instance void BasicProgram.BasicProgram::.ctor()
        IL_0006: stloc.0
        IL_0007: ldloc.0
        IL_0008: ldc.i4.s 10
        IL_000a: callvirt instance void BasicProgram.BasicProgram::set(int32)
        IL_000f: nop
        IL_0010: ret
    } // end of method BasicProgram::Main

    .method public hidebysig specialname rtspecialname
        instance void .ctor () cil managed
    {
        // Method begins at RVA 0x2079
        // Code size 7 (0x7)
        .maxstack 8

        IL_0000: ldarg.0
        IL_0001: call instance void [mscorlib]System.Object::.ctor()
        IL_0006: ret
    } // end of method BasicProgram::.ctor

} // end of class BasicProgram.BasicProgram
```

```
public class BasicProgram {
    int value;

    void set(int value){
        this.value = value;
    }

    public static void main(String[] args){
        BasicProgram bp = new BasicProgram();
        bp.set(10);
    }
}
```

```
Compiled from "BasicProgram.java"
public class BasicProgram {
  int value;
  public BasicProgram();
  void set(int);
  public static void main(java.lang.String[]);
}

Compiled from "BasicProgram.java"
public class BasicProgram {
  int value;

  public BasicProgram();
    Code:
       0: aload_0
       1: invokespecial #10                 // Method java/lang/Object."<init>":()V
       4: return

  void set(int);
    Code:
       0: aload_0
       1: iload_1
       2: putfield      #18                 // Field value:I
       5: return

  public static void main(java.lang.String[]);
    Code:
       0: new           #1                  // class BasicProgram
       3: dup
       4: invokespecial #22                 // Method "<init>":()V
       7: astore_1
       8: aload_1
       9: bipush        10
      11: invokevirtual #23                 // Method set:(I)V
      14: return
}
```

Optimisation et profiling (JVM)

- JVM: réalise des opérations d'optimisation à l'exécution (hotspot):
  - Elle interprète le code
  - Si une méthode est appelée régulièrement elle est compilée en langage machine.
  - Si cette méthode est encore appelée de nombreuses fois une compilation plus lente mais plus optimisée s'opère

  - De nombreuses optimisations hotspot existent et diffèrent d'une VM client à une VM serveur: http://openjdk.java.net/groups/hotspot/docs/HotSpotGlossary.html

Optimisation et profiling (CLR)

- CLR: repose sur les JIT strategy:
  - Le code est entièrement compilé sans optimisation par un JIT rapide.
  - Si on s'aperçoit qu'une partie du code est exécuté souvent on utilise un JIT plus lent qui optimise le code.

Les types

- Java: les types primitifs sont des types particuliers. Il n'héritent pas du type Object. Par exemple on ne peut pas faire: `3.getClass();`

- C#: tous les types héritent de Object et possèdent donc les méthodes associées.

Les types génériques

- Encapsule des opérations non spécifiques à un type de données particulier (Collection, …)
- Exemple:

```
public static void main(String[] args){
    List<String> listString = new ArrayList<>();
    listString.add("Une chaine");
    listString.add("Une autre chaine");

    List<Integer> listInt = new ArrayList<>();
    listInt.add(1);
    listInt.add(2);

    listString.add(3);
}
```

Les types génériques

- Exemple:

```
List<String> listString = new ArrayList<>();
List<Integer> listInt = new ArrayList<>();

boolean answer = listString.getClass() == listInt.getClass();
```

- Que vaut le boolean?

Réponse:

- **Vrai** en Java
- **Faux** en C#

Name	Value
"listString.getClass()"	(java.util.ArrayList) (id=17)
"listInt.getClass()"	(java.util.ArrayList) (id=17)
"answer"	true

Name	Value
listString.GetType()	{Name = "List`1" FullName = "System.Collections.Generic.List`1[[System.String, msco
listInt.GetType()	{Name = "List`1" FullName = "System.Collections.Generic.List`1[[System.Int32, mscorl
answer	false

Les types génériques

- Java:
  - Les types génériques devaient pouvoir s'exécuter sur une VM non modifiée. Tous les génériques sont remplacés par le type **Object** à la compilation

```
List<String> list = new ArrayList<>();
list.add("text");
String value = list.get(0);
```

```
List<Object> list = new ArrayList<>();
list.add("text");
String value = (String) list.get(0);
```

- C#:
  - Chaque type générique possède un code particulier

Inlining

- Optimisation permettant de remplacer l'appel d'une méthode par son contenu:

```
int max(int a, int b){
    return (a > b) ? a : b;
}

void compute(){
    a = max(x, y);
}
```

```
void compute(){
    a = (x > y) ? x : y;
}
```

- Permet d'éviter les sauts durant l'exécution du programme et d'économiser de la place sur la pile des appels

Inlining

- Cette optimisation se fait au moment de l'exécution.
- Les méthodes finales sont plus faciles à inliner que les méthodes virtuelles.
  - Pour inliner une méthode virtuelle il faut chercher dans toutes les classes héritées si la méthode existe.

- En java, par défaut les méthodes sont virtuelles
- En C# elle sont finales.

Exécuter du C++

- Pour exécuter du C++ en Java: JNI (Java Native Interface)
  - Le code C++ se trouve dans des fichiers séparés.
- En C#: blocs de code unsafe.
  - Visible par tous les développeurs

Exécuter du C++ (Java)

Header

```
#include <jni.h>
/* Header for class TestJNI1 */
#ifndef _Included_TestJNI1
#define _Included_TestJNI1

#ifdef __cplusplus
extern " C " {
#endif

JNIEXPORT void JNICALL Java_TestJNI1_afficherBonjour(JNIEnv *, jobject);

#ifdef __cplusplus
}
#endif
#endif
```

C++

```
#include <jni.h>
#include <stdio.h>
#include "TestJNI1.h"

JNIEXPORT void JNICALL
Java_TestJNI1_afficherBonjour(JNIEnv *env, jobject obj){
printf(" Bonjour\n ");
return;
}
```

Java

```
class TestJNI {

    public native void afficherBonjour();
    static {
        System.loadLibrary("mabibjni");
    }

    public static void main(String[] args) {
        TestJNI test = new TestJNI();
        test.afficherBonjour();
    }
}
```

Exécuter du C++ (C#)

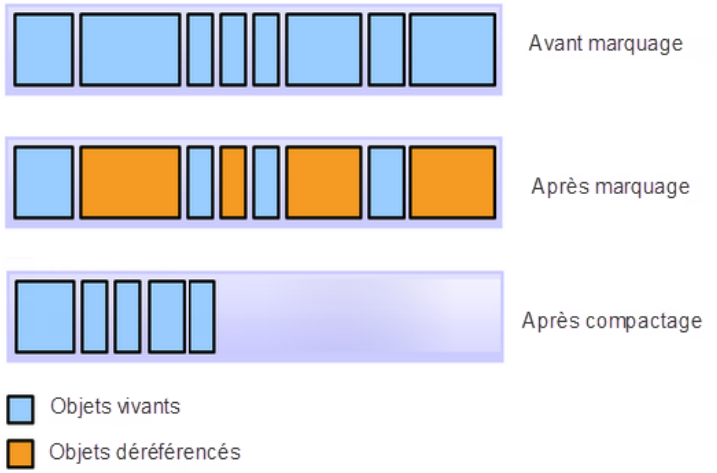
```
class UnsafeTest
{
    [DllImport("msvcrt.dll", CallingConvention = CallingConvention.Cdecl)]
    private static extern void printf(string format);

    unsafe static void afficherBonjour()
    {
        printf("Bonjour\n");
    }

    unsafe static void Main()
    {
        afficherBonjour();
    }
}
```

Garbages collectors

- Le déclenchement:
  - La mémoire du système baisse trop
  - La taille des objets alloués dépasse un seuil
  - La méthode de nettoyage est appelée (System.gc() en Java et GC.Collect() en C#)

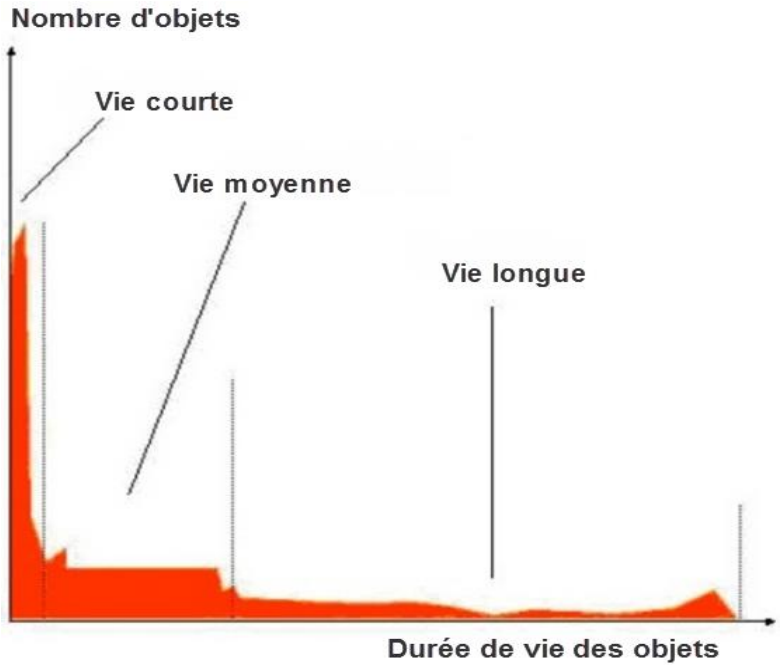
Garbages collectors

- Les étapes:
  - Le marquage: crée une liste des objets référencés
  - La suppression: liste les emplacements ou de nouveaux objets peuvent être alloués.
  - Le compactage déplace les objets qui ont survécu



Garbages collectors

- Durée de vie des objets:
  - On distingue 3 générations d'objets: les objets jeunes, les objets vieux et les objets permanents
  - A chaque nettoyage les objets encore vivants sont déplacés dans la catégorie au dessus



Les exceptions

- Java: checked et unchecked exception
- C#: pas d'exception checked

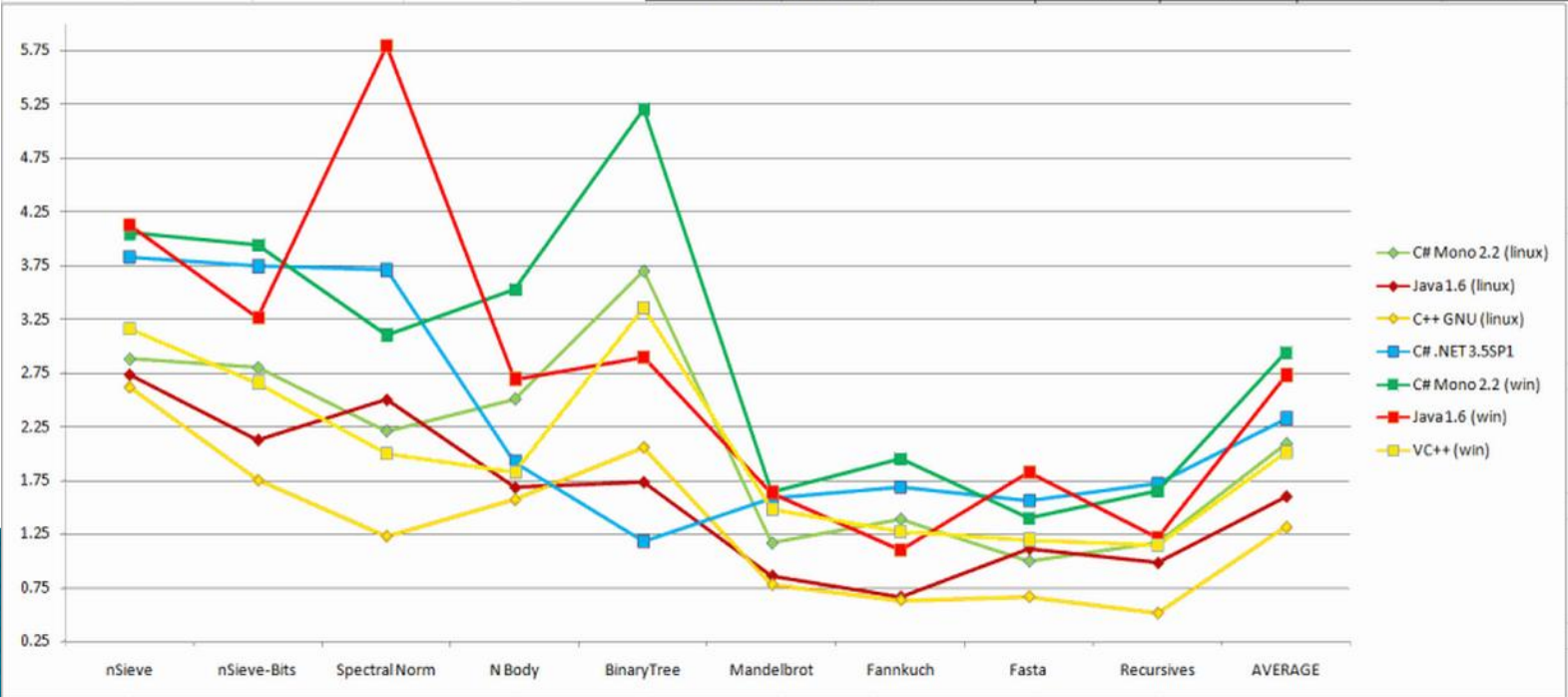
Futures évolutions

- Java 9: Support des heap de plusieurs Giga, auto amélioration de la JVM.
- Java 10: Suppression des types primitifs, tableau de 64 bits adressables

- C#: vers un code compilé

Times in Seconds (lower the number, the better the result)

	Windows Vista Business SP1						openSUSE 11.1				
Benchmark	C# .NET 3.5SP1	C# Mono 2.2 (win)	Java 1.6 (win)	PHP 5.2.8 (win)	Ruby 1.8.6 (win)	VC++ (win)	C# Mono 2.2 (linux)	Java 1.6 (linux)	PHP 5.2.8 (linux)	Ruby 1.8.6 (linux)	C++ GNU (linux)
nSieve	3.820	4.046	4.124	51.443	120.333	3.161	2.874	2.735	47.344	112.271	2.613
nSieve-Bits	3.736	3.933	3.262	97.332	180.218	2.657	2.794	2.127	92.918	168.143	1.746
Spectral Norm	3.701	3.102	5.790	512.355	701.080	1.992	2.203	2.501	494.443	588.907	1.224
N Body	1.921	3.523	2.691	467.333	757.462	1.825	2.502	1.682	433.999	848.357	1.564
BinaryTree	1.182	5.201	2.896	31.503	28.448	3.355	3.694	1.732	31.100	23.896	2.052
Mandelbrot	1.584	1.634	1.633	187.864	722.625	1.472	1.161	0.859	179.935	674.209	0.774
Fannkuch	1.684	1.946	1.101	309.008	356.944	1.264	1.382	0.661	306.526	350.876	0.628
Fasta	1.558	1.395	1.820	239.105	375.392	1.192	0.991	1.112	231.002	369.010	0.661
Recursives	1.721	1.645	1.213	295.557	332.385	1.142	1.168	0.978	272.768	310.115	0.508
AVERAGE	2.323	2.936	2.726	243.500	397.210	2.007	2.086	1.599	232.226	382.865	1.308
					% increase over Windows:		*29.0*	*41.3*	*4.6*	*3.6*	*34.8*



Windows		
.NET is	26.4	% faster than Mono
.NET is	17.3	% faster than Java
.NET is	10382.1	% faster than PHP
.NET is	16999.0	% faster than Ruby
C++ is	15.8	% faster than .NET
C++ is	35.8	% faster than Java
C++ is	46.3	% faster than Mono
Java is	7.7	% faster than Mono
Java is	8834.0	% faster than PHP
Java is	14473.5	% faster than Ruby
Mono is	8193.3	% faster than PHP
Mono is	13428.4	% faster than Ruby

Linux		
C++ is	59.5	% faster than Mono
C++ is	22.2	% faster than Java
Java is	30.5	% faster than Mono
Java is	14427.2	% faster than PHP
Java is	23850.7	% faster than Ruby
Mono is	11035.2	% faster than PHP
Mono is	18258.3	% faster than Ruby

Windows Ranking	ALL
VC++	C++ GNU (Linux)
.NET 3.5SP1	Java 1.6 (Linux)
Java 1.6	Mono (Linux)
Mono 2.2	VC++
PHP 5.2.8	.NET 3.5SP1
Ruby 1.8.6	Java 1.6
	Mono 2.2
Linux Ranking	PHP 5.2.8 (Linux)
C++ GNU	Ruby 1.8.6 (Linux)
Java 1.6	PHP 5.2.8
Mono 2.2	Ruby 1.8.6
PHP 5.2.8	
Ruby 1.8.6	

Démonstrations

Conclusion

- Les 2 langages et leurs VM sont très ressemblantes
- La JVM permet des optimisations impressionnantes!
- Java évolue lentement (mais surement?)
- Utiliser le bon outil:
  - multiplateforme?
  - interface graphique?

Sources

- http://www.artima.com/intv/choices.html
- http://en.wikipedia.org/wiki/Comparison_of_C_Sharp_and_Java
- http://fr.slideshare.net/robertbachmann/c-sharp-java20110404
- http://www.iict.ch/Tcom/team/ELS/presentations/ComparatifJavaCDiese.pdf
- http://fr.slideshare.net/jeffz/why-java-sucks-and-c-rocks-final
- http://stackoverflow.com/questions/453610/javas-virtual-machine-and-clr
- http://fr.slideshare.net/alexcheng1982/the-evolution-of-java
- http://wiki-dot.net/TUHH/files/Evolution_of_CS.pdf
- http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html
- http://msdn.microsoft.com/en-us/library/ee787088(v=vs.110).aspx
- http://reverseblade.blogspot.fr/2009/02/c-versus-c-versus-java-performance.html

Questions?