Java vs C#

Sławomir Kierat i Marcin Mieteń

Uniwersytet Warszawski

10 czerwca 2010

Wstęp historyczny - JAVA

- miejsce powstania: laboratoria Sun Microsystems w Mountin View
- rok powstania: 1995
- pod kierownictwem Jamesa Goslinga
- aktualna wersja stabilna: 6.0.160.70
- stworzony na bazie Smalltalka (maszyna wirtualna, odśmiecanie pamięci) oraz C++ (duża część składni i słów kluczowych)

Rysunek: James Gosling

Wstęp historyczny - C#

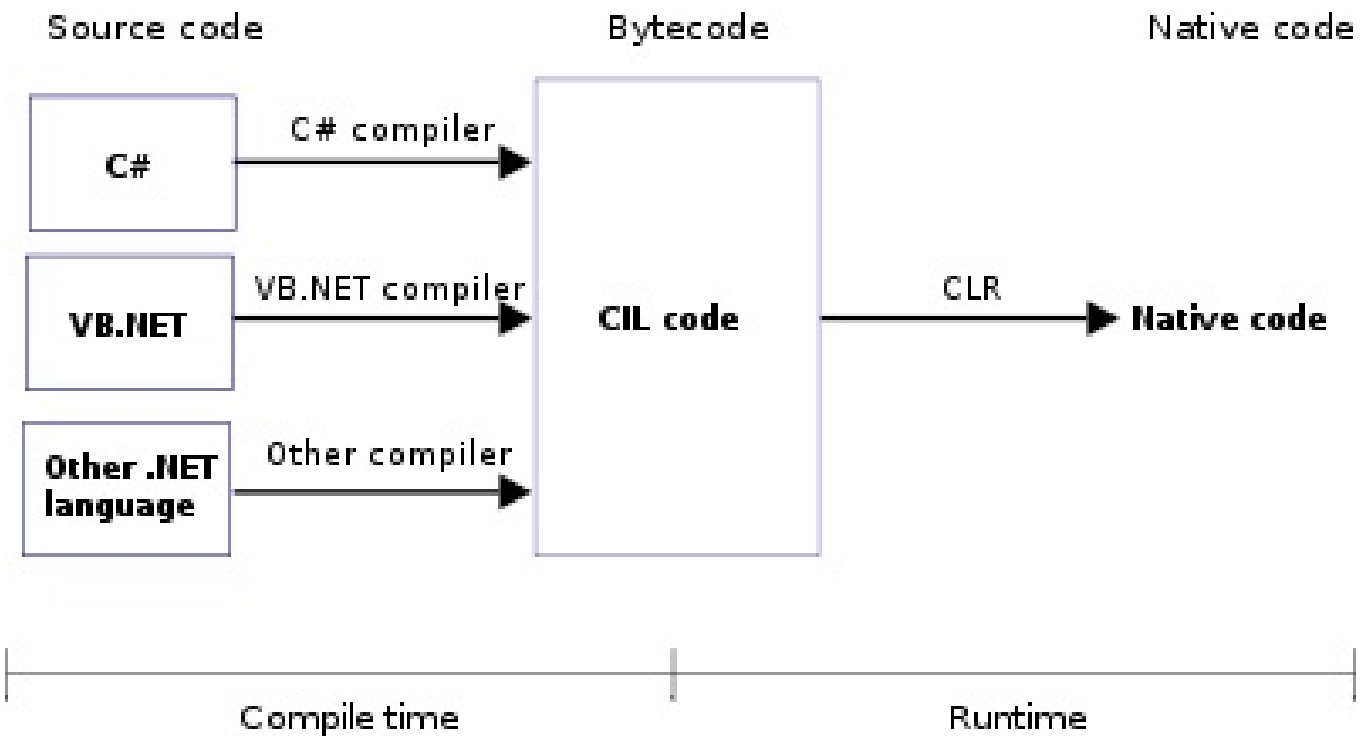
- miejsce powstania: laboratoria Microsoft
- rok powstania: 2000 (wersja C# 1.0)
- pod kierownictwem Andersa Hejlsberga
- aktualna wersja stabilna: C# 3.0
- stworzony na bazie Object Pascala, Delphi, C++ i Java
- wersja 1.0 - rok 2000, 2.0 - rok 2005, 3.0 - rok 2007

Rysunek: Anders Hejlsberg

Common Language Runtime

- Microsoftowa implementacja standardu Common Language Infrastructure
- przewidzione do pracy na wielu:
  - systemach operacyjnych
  - maszynach
- wykonuje kod wyrażony w Common Intermediate Language
- w skład CLR wchodzą:
  - obsługa wykonywania
  - bezpieczeństwo
  - zarządzanie pamięcią
  - Wspólny system typów - CTS

Common Language Runtime



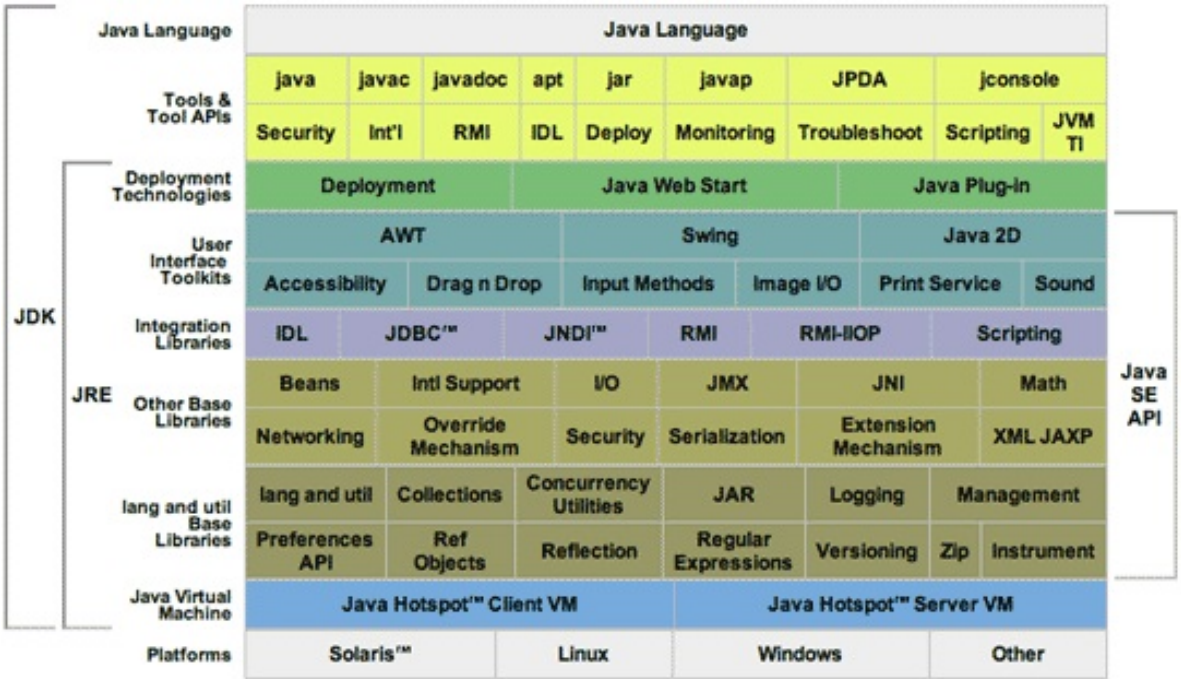
Rysunek: CLR

źródło: http://en.wikipedia.org/wiki/File:CLR_diag.svg

Java Virtual Machine

- wiele implementacji - najpopularniejsza Suna
- wchodzi w skład JRE
- środowisko uruchomieniowe dla programów napisanych w językach:
  - Java
  - Jython
  - JRUBY
  - Scala
  - ADA
- kodem pośrednim jest Java bytecode
- JVM wykonuje pliki w formacie .class .jar
- interpretacja kodu pośredniego spowalnia:
  - przyspieszenie JIT (np Hotspot Sun'a)

Java Platform Standard Edition



Rysunek: JPSE

źródło: http://en.wikipedia.org/wiki/Java_(software_platform)

Składnia

- wspólne pochodzenie z języka C
- wspólne słowa kluczowe (około 40):
  - for, while, if, else, switch, case, void, return
  - new, class, private, public, protected, break, continue
  - abstract, static, this, null, try, catch, volatile
- W Javie a nie w C#
  - assert, boolean, extends, final, implements, instanceof
  - native, package, strictfp, super, synchronized, throws, transient
- W C# a nie w Javie
  - as, base, bool, checked, decimal, delegate, event, explicit
  - extern, fixed, foreach, implicit, internal, lock, namespace
  - object, operator, out, override, params, readonly, ref, is
  - sbyte, sealed, sizeof, stackalloc, string, struct, typeof
  - uint, ulong, unchecked, unsafe, ushort, using, virtual

Szablony i programowanie generyczne

- statyczny polimorfizm czyli szablony:
  - np. List<T>, Dictionary<K,T>
- od .NET 2.0 typy generyczne (dot. głównie kolekcji)
  - przestrzeń nazw System.Collections.Generic
  - przyspieszenie działania - oszczędność na rzutowaniu
  - wykrywanie w czasie kompilacji błędów typu wkładanie obiektów różnych typów do kolekcji
  - w przeciwieństwie do typów uniwersalnych wymagają podania typów w nawiasach "<>"
- odmienne droga do programowania generycznego:
  - Java - konstrukcja językowa, dotyczy tylko kompilatora
  - C# - dodatkowo JIT dokonuje weryfikacji podczas ładowania

Obiektowość

- wszystko jest obiektem
- polimorfizm i metody wirtualne
  - W Javie domyślnie metody są wirtualne z wyjątkiem
    - metod statycznych
    - metod ze specyfikatorem final
    - metod prywatnych
  - W C# domyślnie metody nie są wirtualne
    - słowo kluczowe: virtual - metody wirtualne.
    - słowo kluczowe: override - zasłanianie metody klasy bazowej.
    - słowo kluczowe: base - odwołanie do przesłoniętej metody.
- brak metod globalnych
- brak wielodziedziczenia
- mechanizm interfejsów - różnice składniowe

Garbage Collector

feature	Java	C#
Garbage collection	yes	yes
Deterministic disposal	no	yes
Weak references	yes	yes
Soft references	yes	no
Object pinning	no	yes

źródło: http://en.wikipedia.org/wiki/Comparison_of_Java_and_C_Sharp

```
1.    var dr = new DisposableResource();
2.    try
3.    {
4.        ...
5.    }
6.    finally
7.    {
8.        dr.Dispose();
9.    }
```

Wersja skrócona

```
1.    using (var dr = new DisposableResource())
2.    {
3.        ...
4.    }
```

pinned Object

```
public PinnedObject()
{
  handle = GCHandle.Alloc(managedObject, GCHandleType.Pinned);
  ptr = handle.AddrOfPinnedObject();
}

~PinnedObject()
{
  Dispose();
}

public void Dispose()
{
  if (!disposed)
  {
    handle.Free();
    ptr = IntPtr.Zero;
    disposed = true;
  }
}
```

Dokumentacja kodu

Java

```
/** metoda main
 *  @param args tablica napisów
 */
public static void main(String [] args)
```

C#

```
/// <summary> metoda main </summary>
/// <param name="args">tablica napisów</param>
public static int Main(String [] args)
```

Struktury

- "lekka" klasa
- każda struktura niejawnie dziedziczy po System.ValueType
- nie dziedziczy po klasach, ani nie jest przedmiotem dziedziczenia
- nie może zawierać niestandardowego bezparametrowego konstruktora
- nie może zawierać destruktora
- zaleta: szybszy dostęp (stos)
- wada: możliwe tylko przekazywanie przez wartość
- atrybut StructLayout umożliwia określenie rozmieszczenia struktury w pamięci

Wskaźniki

- ze względu na Garbage Collector używanie wskaźników nie jest bezpieczne
- słowo kluczowe **unsafe** dla metod działających na wskaźnikach
- słowo kluczowe **fixed** zapewnia poprawne wskazywanie wskaźnika

```
fixed ( int* p = &pt.x )
{*p = 1;}
public static unsafe void Swap(int* a, int*b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}
```

Referencje

- możliwość przekazywania przez referencje typów prostych
- Dwa możliwe identyfikatory:
  - **out** - parametr inicjalizowany przez metodę wywoływaną
  - **ref** - parametr inicjalizowany przez metodę wywołującą
- Rozróżnienie wpływa na czytelność kodu

```
public static void ChangeMe(out string s)
   { s = "Changed";     }
public static void Swap(ref int x, ref int y)
   { int z = x;    x = y;    y = z;    }
```

- Specjalna konstrukcja dla właściwości w C#

```
class Woman{
    private int age;
    public int Age
    {
        get { return age - 10; }
        set { age = value; }
    }
}
w.Age = 35;
Console.Writeln( w.Age );
```

- umożliwia generowanie bardziej efektywnego kodu
- wymagany jest jeden z parametrów set lub get

Indeksery - właściwości sparametryzowane

```
public class Stack
{
    public object this[int index]
    {
        get { return GetNode(index).Value; }
        set { GetNode(index).Value = value; }
    }
    ...
}
Stack s= new Stack();
...
s[0]=10;
Console.WriteLine(s[0]);
```

Sekcja checked

```
class TestClass
{
   const int x = 2147483647;   // Max int
   const int y = 2;

   public int MethodDef()
   {
      // Using default overflow checking:
      int z = x * y;
      return z;   // Compiler error
   }
}
```

Sekcja unchecked

```
class TestClass
{
   const int x = 2147483647;   // Max int
   const int y = 2;

   public int MethodUnCh()
   {
      // Unchecked statement:
      unchecked
      {
         int z = x * y;
         return z;   // Returns -2
      }
   }
}
```

Regiony kodu

```
namespace ConsoleApplication1
{
    spójny region
}
```

```
namespace ConsoleApplication1
{
    #region spójny region

    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("duqa");
            Console.ReadKey();
        }
    }
    #endregion
}
```

Rysunek: Regiony kodu

Przeciążanie operatorów

```
class MyNumber
{
   private int value;
   public MyNumber(int value)
   {
      this.value = value;
   }
   public static MyNumber operator+(MyNumber n1,MyNumber n2)
   {
      return new MyNumber(n1.value + n2.value);
   }
   ...
}
```

Delegacje

```
public delegate void MyString(string s);
public static void PrintLower(string s){
   Console.WriteLine(s.ToLower());
}
public static void PrintUpper(string s){
   Console.WriteLine(s.ToUpper());
}
    public static void Main()
    {
        MyString myDel;
        myDel = new MyString(PrintLower);
        myDel += new MyString(PrintUpper);
        myDel("Ala ma kota Zbycha.");
   }
}
```

Praca ze zdarzeniami wbudowanymi

- predefiniowana delegacja MouseEventHandler
- predefiniowane zdarzenie MouseDown

```
...
this.MouseDown += new MouseEventHandler(OnMouseDown);
...
private void OnMouseDown(object sender, MouseEventArgs e)
{
    lastX = e.X;
    lastY = e.Y;
}
```

Własne zdarzenia

```
public class IOMonitor
{
   public delegate void IODelegate(String s);
   public event IODelegate DataReceived;
   public void FireReceivedEvent (string msg)
   {
      if (DataReceived != null)
         {DataReceived(msg);}
   }
}
```

Language Integrated Query

- Wykonywanie zapytań do różnego rodzaju kolekcji
  - bazy danych SQL
  - XML
  - zwykłe kolekcje
- Możliwa wstępna optymalizacja zapytań po stronie klienta
- PLinq - .NET 4.0 (Beta)
  - zrównoleglenie wykonywania zapytań

```
string[] names ={"Burke","Connor","Frank","Everett","Albert"};
IEnumerable<string> query = from s in names
                            where s.Length == 5
                            orderby s
                            select s.ToUpper();
foreach (var item in query)
    Console.WriteLine(item);
```

Linq2Sql

- mechanizm relacyjno-obiektowy
  - Tabela $\rightarrow$ Klasa
  - Rekord $\rightarrow$ Obiekt
  - Kolumna $\rightarrow$ Właściwość klasy
  - Procedura składowa $\rightarrow$ Metoda
- prosty edytor (drag and drop)
- bogaty edytor - możliwe modyfikacje modelu
  - nazwy, połączenia
  - dziedziczenia
  - dodefiniowywania klas (klasy partial)
  - jawne użycie SQL

Więcej o Linq na kole .NET, które odbędzie się we wtorek, 5 stycznia 2010 r., o godzinie 16:15 w sali 5070.

Słowo kluczowe var

Deklaracja zmiennej niewiadomego typu.

```
var hColl = new HashSet<string>();
hColl.Add("one");
```

Konstruktory kolekcji

Konstruktory kolekcji z możliwością instancjonowania konkretnymi danymi.

```
var hColl = new HashSet<string>() { "one", "two" };
```

Konstruktory z listą właściwości

```
public class MojaKlasa
{
      public int x;
      public double y;
}
var ob = new MojaKlasa { x=2, y=3.4 };
```

Konstruktory z listą właściwości wywołują:

1. konstruktor bezparametrowy (o ile istnieje)
2. inicjalizacje zmiennych podanych w { } (zmienne musza być publiczne lub musi istnieć dla nich konstrukcja właściwości set)

Typy anonimowe

Poniższy zapis wygeneruje klasę zawierającą podane na liście parametrów zmienne oraz właściwości do zapisu i odczytu tych zmiennych.

```
var ob = new { x=5, y=5.6, c=''napis''};
```

Rozszerzenia klas

Istnieje mechanizm pozwalający dodawać metody bez ingerencji w kod danej klasy. Należy zaimportować namespace z specjalną klasą spełniającą określone warunki:

- klasa musi być statyczna
- metoda w takiej klasy odpowiada dodaniu metody do innej klasy o ile:
  - metoda ta jest statyczna
  - pierwszy argument jest typu rozszerzanej klasy
  - pierwszy argument posiada dodatkowo słowo kluczowe **this**

Rozszerzenia klas

```
public static class Extensions
{
  public static int silnia(this int i)
  {
    if (i == 1) return 1;
    else return (i - 1).silnia() * i;
  }
}

var a = 3.silnia(); // da nam a=6
```

Partial methods

Mechanizm pozwalający definiować klasę w dwóch plikach. Często jedna z części jest generowana automatycznie (np. z modelu bazy danych) a drugą część pisze programista nie ingerując w wygenerowaną część klasy.

```
public partial class Employee{
 //Ta metoda musi być zdefiniowna w innej części
  partial void doSomething();
  public void doWork(){ this.doSomething; }
}
public partial class Employee{
  partial void doSomething(){...}
  public void myFunction(){this.doWork() }
}
```

Lambda expressions

Mechanizm ten to rozszerzenie mechanizmu anonimowych delegatów. Kod:

```
var Asia = list2.Find(delegate(Person p)
  {
    return p.Name == "Asia";
  });
```

Można zastąpić poniższym lambda wyrażeniem:

```
var Asia = list2.Find(p => p.Name == "Asia");
```

Klasy anonimowe

- Java posiada klasy anonimowe
- C# NIE ma klas anonimowych.
- Większość zastosowań klas anonimowych w C# zastępują funkcje anonimowe (delegate).

```
new KlasaBadzInterfejsBazowy([lista argumentów])
{
  treść klasy anonimowej
}
```

Klasyczne użycie klasy anonimowej

```
Button b = new Button("przycisk");
    b.addActionListener(
      new ActionListener() {
        public void actionPerformed(ActionEvent e) {
          // obsługa zdarzenia e
        }
      }
    );
```

W C#

```
b.Click += delegate { // obsługa wciśnięcia przycisku };
```

Obsługa wyjątków

- W Javie i C# obsługa wyjątków jest bardzo podobna (blok try{}catch(...){}final{} , throw new Exception()).
- Istotna różnica: C# nie ma wyjątków kontrolowanych.
- W Javie istnieją dwa rodzaje wyjątków:
  - wyjątki kontrolowane (ang. checked exceptions) rozszerzające Exception - programista pisząc metodę zobowiązany jest przechwycić wszystkie możliwe kontrolowane wyjątki lub zdefiniować jakie kontrolowane wyjątki może zgłosić metoda (w nagłówku metody używamy *throws*).
  - wyjątki niekontrolowane (ang. unchecked exceptions) rozszerzające RuntimeException - nieprzechwycone wyjątki niekontrolowane (także podklasy Throwable ale nie Exception).

Wyjątki kontrolowane

- Dzięki wyjątkom kontrolowanym programista zawsze wie jakie wyjątki może zgłosić metoda.
- W C#:
  - o ile twórcy biblioteki nie udokumentował wyjątków, nie wiemy które z nich powinny być przechwycone.
  - dodatkowo po prostu łatwiej możemy zapomnieć o obsłużeniu jakiegoś wyjątku.

Inne cechy wyróżniające Javę

- Wieloplatformowość
  - **Java:** Linux, Windows, Solaris, OS/2, AIX, MacOS, FreeBDS
  - **C#:** Windows, Windows, Windows, Linux (Mono), MacOS (Mono)
- strictfp - obliczenia w metodzie lub klasie oznaczonej w ten sposób są zgodne ze standardem IEEE-754.
- interfejsy mogą posiadać stałe.

Nowości zapowiadane w Java 7

- Moduły (nowe słowo kluczowe module, możliwość tworzenia zależności między modułami)
- Switch po Stringu (aktualnie tylko byte, short, char, int i enumerated types)
- catch (Exception1 | Exception2 | Exception3 e)
- Map<String, List<String> > anagrams = new HashMap<>();
- Często w kodzie sprawdzamy, czy obiekt jest nullem i wtedy robimy coś innego. Tutaj z pomocą ma przyjść Null-Safe Operator(znak ?):
  String s = object?.toString() ?: "nothing";
- Plan wydania -> Marzec 2010

Pakiety vs Przestrzenie nazw

Java

- Podział zależny od systemu plików.
  pl.example.Person => pl/example/Person.java
- Person.java => public class Person {...} + dowolna liczba klas niepublicznych.
- Zalecana konwencja ustalania nazw zgodnie z domeną (aby zachować unikatowość).
- + Przy zachowaniu konwencji nie ma problemów z dołączaniem nowych bibliotek.
- - Przy zachowaniu konwencji często mamy długą ścieżkę do docelowego katalogu z klasami (wsparcie w IDE do omijania zbędnych katalogów)

Pakiety vs Przestrzenie nazw

C#

- Podział niezależny od systemu plików.
- W każdym pliku możemy definiować dowolną liczbę klas.
- Można definiować jedną klasę w wielu plikach (słowo kluczowe partial)
- + Większa możliwość posegregowania plików (możemy mieć w różnych katalogach pliki z klasami należącymi do tej samej przestrzeni nazw)
- - Możliwe problemy z ładowaniem bibliotek z uwagi na brak konwencji przestrzeni nazw (ale są mechanizmy aliasowania ładowanych bibliotek)

Pakiety a przestrzenie nazw - składnia

Java

```
package com.example.package;
import com.example.package_imported.*;
import com.example2.Person;
// Importowanie pól statycznych danej klasy
import static java.awt.Color.*;
public class A
{
//constants not qualified thanks to static import
System.out.println(RED + " plus " + YELLOW + " is " + ORANGE
}
```

Pakiety a przestrzenie nazw

C#

```
using example.namespace.name1;
using MOJ_ALIAS = example.namespace.name2;
using IntList = System.Collections.Generic.List<int>
namespace examle.ns.name
{
    class A
    {

    }
}
```

Importowanie - class i public class

Java

class NAZWA_KLASY {...} widoczna tylko w swoim pakiecie
public class NAZWA_KLASY {...} dostęp do klasy z dowolnego pakietu

C#

class NAZWA_KLASY {...}
Czyni klasę dostępną z innych klas z tej samej przestrzeni
public class NAZWA_KLASY { ... }
Umożliwia dostęp do klasy dla innych przestrzeni

Metody Wirtualne

Java

- metody statyczne NIE są wirtualne
- metody niestatyczne są zawsze wirtualne
- nie można stworzyć publicznych niestatycznych i niewirtualnych metod!

C#

- metody wirtualne tak jak w C++.
- metody domyślnie są niewirtualne.
- aby metoda była wirtualna dopisujemy słowo kluczowe **virtual**.
- aby przedefiniować metodę wirtualną dodajemy słowo kluczowe **override**.

Metody Wirtualne w C#

```
class A{
        public virtual void test()
        {
            System.Console.WriteLine("A");
        }
    }
    class B : A{
        public override void test()
        {
            base.test();
            System.Console.WriteLine("B");
        }
    }
```

IDE

Java

- Eclipse
- NetBeans
- Borland JBuilder
- IntelliJIDEA

C#

- Microsoft Visual Studio, IDE dla C#
- C#Builder
- SharpDevelop - open-source C# IDE dla Windows
- MonoDevelop - open-source C# IDE dla Linux, Windows i Mac OS X

Podsumowanie

Java

- + wieloplatformowa
- + lepiej przetestowana
- + duże wsparcie społeczności
- - powolny rozwój i brak istotnych zmian ułatwiających programowanie.

C#

- + wyposażony w dużo większą liczbę mechanizmów.
- + dynamiczny rozwój języka
- - głównie na Windows.
- - mnogość mechanizmów => dłuższy czas nauki języka i większa komplikacja kodu.