Modelowanie i Analiza Systemów informacyjnych (MAS)

dr inż. Mariusz Trzaska
mtrzaska@pjwstk.edu.pl

Wykład 3

Wybrane konstrukcje obiektowych języków programowania (2)

- Kontynuacja poprzedniego wykładu

Wykorzystano materiały z Thinking in Java (3rd edition) autorstwa Bruce'a Eckel'a

http://www.mindview.net/Books/TIJ

Przechowywanie wielu elementów

- Zastosowanie
- Tablica
- Pojemniki
  - Kolekcje
  - Mapy
- Zalety i Wady

Tablice

- Zalety
- Wady

```
int[] a1 = { 1, 2, 3, 4, 5 };
int[] a2 = a1; // co tu się dzieje?

Integer[] a = new Integer[rand.nextInt(20)];
a[i] = new Integer(rand.nextInt(500));
```

- Użyteczne metody: `java.util.Arrays`

Kolekcje

- List: kolejność
  - ArrayList: szybki dostęp do elementów o określonym indeksie, wolne wstawianie w środku

```
Collection c = new ArrayList();
for(int i = 0; i < 10; i++)
    c.add(Integer.toString(i));

Iterator it = c.iterator();
while(it.hasNext())
    System.out.println(it.next());
```

  - LinkedList: bardzo szybkie wstawianie i usuwanie oraz optymalna szybkość dostępu

Kolekcje (2)

- Set: brak kolejności oraz powtórzeń
  - HashSet: szybkie wyszukiwanie
  - TreeSet
- Funkcjonalność
  - `boolean add(Object)`
  - `boolean addAll(Collection)`
  - `void clear( )`
  - `boolean contains(Object)`
  - `boolean containsAll(Collection)`
  - `boolean isEmpty( )`

Kolekcje (3)

- Funkcjonalność – c. d.
  - `Iterator iterator( )`
  - `boolean remove(Object)`
  - `boolean removeAll(Collection)`
  - `boolean retainAll(Collection)`
  - `int size( )`
  - `Object[]toArray( )`
  - `Object[]toArray(Object[] a)`

Mapy

- Przechowuje: klucz → wartość
- Funkcja hash'ująca oraz equals
- Różne rodzaje
  - HashMap (bazuje na hashtable)
  - TreeMap (drzewa czerwono-czarne)
- Wady i zalety

Pojemniki - wydajność

- List (mniej = lepiej)

Type	Get	Iteration	Insert	Remove
array	172	516	na	na
ArrayList	281	1375	328	30484
LinkedList	5828	1047	109	16
Vector	422	1890	360	30781

Pojemniki – wydajność (2)

- Set (mniej = lepiej)

Type	Test size	Add	Contains	Iteration
TreeSet	10	25.0	23.4	39.1
	100	17.2	27.5	45.9
	1000	26.0	30.2	9.0
HashSet	10	18.7	17.2	64.1
	100	17.2	19.1	65.2
	1000	8.8	16.6	12.8
LinkedHashSet	10	20.3	18.7	64.1
	100	18.6	19.5	49.2
	1000	10.0	16.3	10.0

Pojemniki – wydajność (3)

- Map (mniej = lepiej)

Type	Test size	Put	Get	Iteration
TreeMap	10	26.6	20.3	43.7
	100	34.1	27.2	45.8
	1000	27.8	29.3	8.8
HashMap	10	21.9	18.8	60.9
	100	21.9	18.6	63.3
	1000	11.5	18.8	12.3
LinkedHashMap	10	23.4	18.8	59.4
	100	24.2	19.5	47.8
	1000	12.3	19.0	9.2
IdentityHashMap	10	20.3	25.0	71.9
	100	19.7	25.9	56.7
	1000	13.1	24.3	10.9
Hashtable	10	18.8	18.7	65.7
	100	19.4	20.9	55.3
	1000	13.1	19.9	10.8

Kolekcje – użyteczne funkcje

- Klasa Collections:
  - `max(Collection)`
  - `min(Collection)`
  - `max(Collection, Comparator)`
  - `min(Collection, Comparator)`
  - `indexOfSubList(List source, List target)`
  - `lastIndexOfSubList(List source, List target)`
  - `replaceAll(List list, Object oldVal, Object newVal)`
  - `reverse( )`
  - `rotate(List list, int distance)`
  - `copy(List dest, List src)`
  - `swap(List list, int i, int j)`
  - `fill(List list, Object o)`
  - `nCopies(int n, Object o)`
  - `list(Enumeration e)`

Java Generics

- Co przechowujemy w pojemniku?
- Co jest nie tak w poniższym programie?

```
private void testCollection() {
  List list = new ArrayList();
  list.add(new String("Good bye!"));
  list.add(new Integer(95));
  printCollection(list);
}

private void printCollection(Collection c) {
  Iterator i = c.iterator();
  while(i.hasNext()) {
    String item = (String) i.next();
    System.out.println("Item: "+item);
  }
}
```

Java Generics (2)

- Rozwiązanie: klasy parametryzowane (generics)

```
private void testCollection() {
    List<String> list = new ArrayList<String>();
    list.add(new String("Hello world!"));
    list.add(new String("Good bye!"));
    list.add(new Integer(95));
    printCollection(list);
  }

  private void printCollection(Collection c) {
    Iterator<String> i = c.iterator();
    while(i.hasNext()) {
      String item = i.next();
      System.out.println("Item: "+item);
    }
  }
```

Nowa pętla for

- Rozwiązanie klasyczne

```
public void oldFor(Collection c) {
 for(Iterator i = c.iterator(); i.hasNtext();) {
    String str = (String) i.next();
    sb.append(str);
  }
}
```

- Nowe rozwiązanie

```
public void newFor(Collection<String> c) {
  for(String str : c) {
    sb.append(str);
  }
}
```

System we/wyj

- Strumień wejściowy
  - Tablica bajtów
  - Obiekt typu String
  - Plik
  - Sekwencja innych strumieni
  - Inne, np. połączenie sieciowe
- Strumień wyjściowy
- Reader'y
- Writer'y

System we/wyj (2)

- Czytanie pliku linia po linii

```
BufferedReader in = new BufferedReader(
  new FileReader("IOStreamDemo.java"));
String s, s2 = new String();
while((s = in.readLine())!= null)
  s2 += s + "\n";
in.close();
```

- Odczyt danych z pamięci

```
StringReader in2 = new StringReader(s2);
int c;
while((c = in2.read()) != -1)
  System.out.print((char)c);
```

System we/wyj (3)

- Zapis do pliku linia po linii

```
try {
     BufferedReader in4 = new BufferedReader(new
StringReader(s2));
     PrintWriter out1 = new PrintWriter(new BufferedWriter(new
FileWriter("IODemo.out")));
     int lineCount = 1;
     while((s = in4.readLine()) != null )
        out1.println(lineCount++ + ": " + s);
      out1.close();
} catch(EOFException e) {
    System.err.println("End of stream");
}
```

System we/wyj (4)

- Zapis i odczyt danych do/z pliku

```
try {
      DataOutputStream out2 = new DataOutputStream(new BufferedOutputStream(
          new FileOutputStream("Data.txt")));
      out2.writeDouble(3.14159);
      out2.writeUTF("That was pi");
      out2.writeDouble(1.41413);
      out2.writeUTF("Square root of 2");
      out2.close();

      DataInputStream in5 = new DataInputStream(new BufferedInputStream(
          new FileInputStream("Data.txt")));
      System.out.println(in5.readDouble());
      System.out.println(in5.readUTF());
      System.out.println(in5.readDouble());
      System.out.println(in5.readUTF());
} catch(EOFException e) {
      throw new RuntimeException(e);
}
```

System we/wyj (5)

- Przekierowanie standardowego wej/wyj
  - `System.setIn(InputStream)`
  - `System.setOut(PrintStream)`
  - `System.setErr(PrintStream)`

```
PrintStream out = new PrintStream(
new BufferedOutputStream(
new FileOutputStream("test.out")));
System.setOut(out);
```

System we/wyj (6)

- Stare i nowe API:
  - java.io.*
  - java.nio.* (szybkość)
- Pliki Memory-mapped
- Blokowanie plików (locking)
- Kompresja
  - ZipOutputStream
  - GZIPOutputStream
  - ZipInputStream
  - GZIPInputStream
- Pliki JAR
- Serializacja

System we/wyj – klasa File

- java.io.File
  - Wyszukiwanie plików
  - Informacje o plikach
    - Atrybuty,
    - Lokalizacja.
  - Usuwanie plików
  - Tworzenie katalogów
  - Usuwanie katalogów
  - Zmiana nazwy
  - pathSeparator

Implementacja GUI - uwagi

- Dodawanie obiektów, a nie String'ów do widgetów - przesłonięcie metody toString(). Dzięki temu nie trzeba później wyszukiwać obiektów na podstawie indeksu.
- Opakowywanie kodu biznesowego w metody, a nie umieszczanie go bezpośrednio w metodach obsługi zdarzeń.

Implementacja GUI – uwagi (2)

- Każda kontrolka ma własnego listener'a (niewspółdzielenie listener'ów bez dobrego uzasadnienia).
- Korzystanie ze zdarzeń/słuchaczy „logicznych" (np. actionPerformed), a nie sprzętowych (i.e. MouseListener).

Java JDK 7 – co nowego?

- Wsparcie dla dynamicznie typowanych języków;
- Typ string w konstrukcji switch;
- Nowy konstrukcja try-with-resources i wielokrotne catch;
- Wnioskowanie o typach – notacja diamentowa <>;
- Uproszczone wywoływanie metod z wieloma parametrami;

Java JDK 7 – co nowego? (2)

- Ulepszone wsparcie dla kolekcji (w tym wielowątkowych);
- Wsparcie dla Unicode 6.0;
- Nowe API dla I/O (pliki, socket'y);
- JDBC 4.1;
- Ulepszenia w Java2D;
- Nowy L&F: Nimbus;
- Nowe możliwości w zakresie syntezy dźwięku (Gervill);

Java JDK 7 – co nowego? (3)

- Ulepszona obsługa XML;
- Propozycje dla wersji JDK 8 lub późniejszych:
  - Ulepszone adnotacje;
  - Ulepszone zarządzanie kolekcjami;
  - Projekt Lambda;
  - Modularyzacja;

Więcej na: http://openjdk.java.net/projects/jdk7/features/

Java JDK 8 – co nowego?

- Metody klasowe (static) w interfejsach.

```
public static <T extends Comparable<? super T>>
Comparator<T> naturalOrder() {
    return (Comparator<T>)
        Comparators.NaturalOrderComparator.INSTANCE;
}
```

Źródło: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/

Java JDK 8 – co nowego? (2)

- Domyślna implementacja metod (`default`) w interfejsach (ale nie dla klasy `Object`). Również w MS C# 8.0.

```
public default void forEach(Consumer<? super T> action) {
    Objects.requireNonNull(action);
    for (T t : this) {
        action.accept(t);
    }
}
```

- Interfejsy funkcyjne (zawierają tylko jedną metodę). Przydatne dla wyrażeń lambda, referencji do metod lub konstruktorów.

Źródło: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/

Java JDK 8 – co nowego? (3)

- Wyrażenia Lambda
  - Funkcjonalność jako parametr metody;
  - Ich kompatybilność jest definiowana przy pomocy typów wejściowych, wyjściowych i wyjątków.

```
Comparator<String> c = (a, b) -> Integer.compare(a.length(),
                                                 b.length());
```

  - Mogą korzystać z zewnętrznych parametrów (zmiennych), ale tylko niezmiennych (m.in. zadeklarowanych jako final).

Źródło: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/

Java JDK 8 – co nowego? (4)

- Nowy funkcyjne interfejsy w java.util.function.
  - Strumień może być przetworzony tylko raz;
  - Sekwencyjne lub równoległe;
  - „Fluent” API

```
int sumOfWeights = blocks.stream().filter(b -> b.getColor() == RED)
                                  .mapToInt(b -> b.getWeight())
                                  .sum();
```

Źródło: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/

Java JDK 8 – co nowego? (5)

- Nowy funkcyjne interfejsy – c.d.
  - Pośrednie operacje z „leniwą” ewaluacją:
    - filter, map, flatMap, peek, distinct, sorted, limit, substream,
  - Operacje kończące:
    - forEach, toArray, reduce, collect, min, max, count, anyMatch, allMatch, noneMatch, findFirst, findAny

Źródło: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/

Java JDK 8 – co nowego? (6)

- Nowy funkcyjne interfejsy – c.d.
  - Strumienie obiektowe oraz „prymitywne”

```
List<String> strings = Arrays.asList("a", "b", "c");
strings.stream()                          // Stream<String>
       .mapToInt(String::length)          // IntStream
       .longs()                           // LongStream
       .mapToDouble(x -> x / 10.0)        // DoubleStream
       .boxed()                           // Stream<Double>
       .mapToLong(x -> 1L)                // LongStream
       .mapToObj(x -> "")                 // Stream<String>
       ...
```

Źródło: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/

Java JDK 8 – co nowego? (7)

- Odniesienia do metod.
- Nowy funkcyjne interfejsy w java.util.function.
- Ulepszenia we wnioskowaniu o typach (notacja <>).
- Nowe klasy do zarządzania czasem/datami: pakiet java.time (niezmienne wartości).
  - `LocalDate`, `LocalDateTime`, `Period`,
  - wygodne tworzenie instancji,
  - operacje na datach.

https://www.baeldung.com/java-8-date-time-intro

Java JDK 8 – co nowego? (8)

- Wsparcie dla przetwarzania strumieniowego wartości w API IO/NIO.

```
try (Stream lines = Files.lines(path, UTF_8)) {
    lines.onClose(() -> System.out.println("done"))
            .forEach(System.out::println);
}
```

- Zmiany w refleksji i adnotacjach.
- Nowy silnik dla JavaScript'u (Nashorn).

Więcej: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/, http://www.infoq.com/news/2013/08/everything-about-java-8

Java JDK 8 – co nowego? (9)

- Nowe metody dodane do API kolekcji (używając domyślnej implementacji metod w interfejsach).
- Usprawnienia w API dla programowania równoległego.

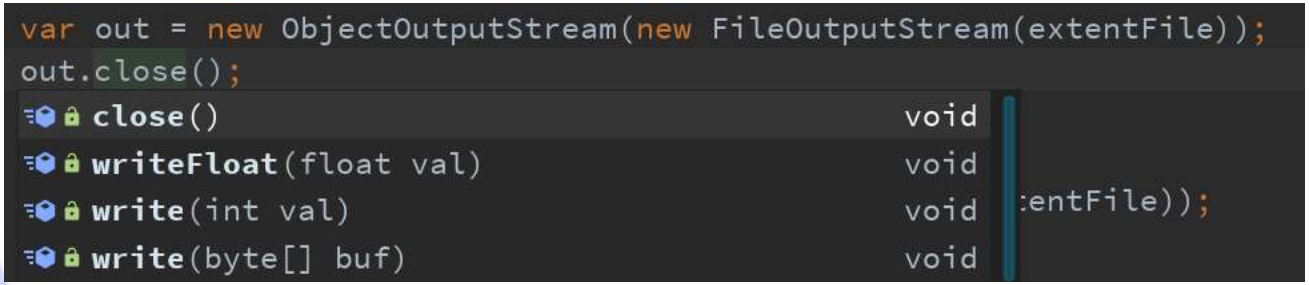
Więcej: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/, http://www.infoq.com/news/2013/08/everything-about-java-8

Java JDK 10 – co nowego? (10)

- Słowo kluczowe var (analogicznie jak w MS C#):
  - mocne typowanie na podstawie typu wartości podstawianej (Local variable type inference)

```
var out = new ObjectOutputStream(new FileOutputStream(extentFile));
out.close();
```

  - podpowiadanie przez IDE



Więcej informacji: *https://stackify.com/whats-new-in-java-10/*

Wydajność języków programowania

- Który z języków programowania jest szybszy?
  - Java?
  - MS C#?
  - MS C++?
- Czy Java jest rzeczywiście dużo wolniejsza od C++?
- Istnieje wiele różnych opinii na ten temat…
- Najlepiej samemu to sprawdzić!

Testy wydajności

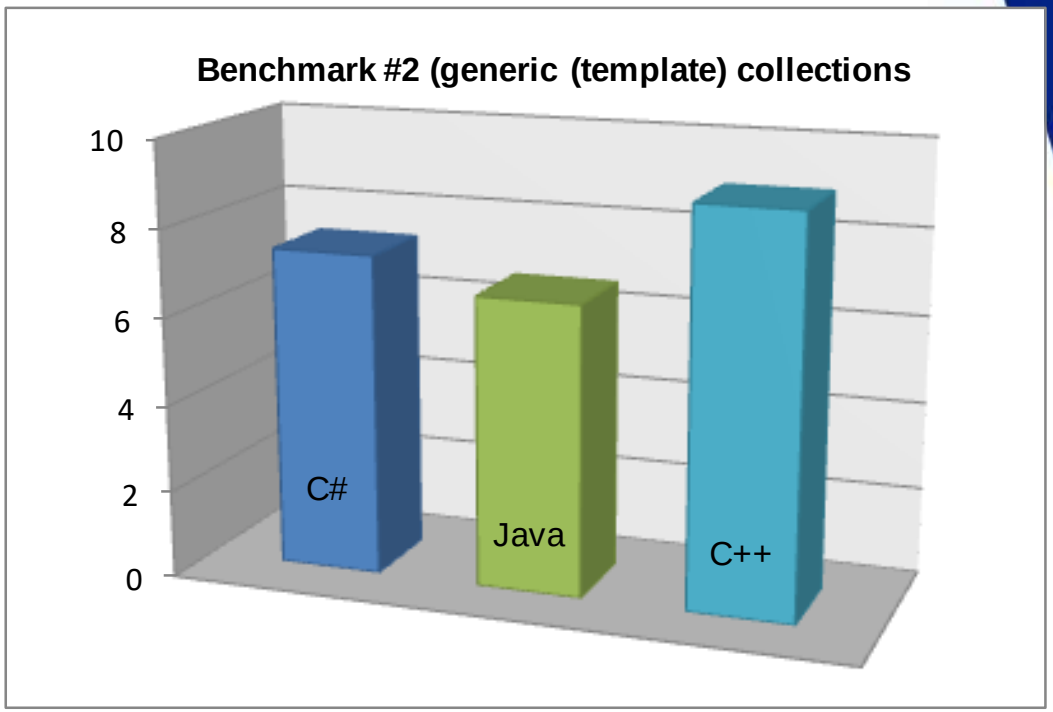
- Prosty test: http://www.mtrzaska.com/plik/rozne/simple-performance-tests-java-vs-c-vs-c-sources
  mtrzaska.com → Pliki → Różne → „Simple performance tests: Java vs C# vs C++”
- Dokładny opis oraz pełen kod źródłowy.
- Środowisko
  - Java: jdk-1_6_0_16
  - MS C#: MS VS 2008SP1 Express, .NET 3.5SP1 (console app)
  - MS C++ 2008 (console app)
  - Windows 7 x64, 4 GB RAM, Intel Core2 Quad Q9550 (2,83 GHz)

Testy wydajności (2)

- Test #1
  - Sprawdza szybkość działania kolekcji mapujących (Hashtable dla C#, HashMap dla Java).
  - Najpierw dodano 5 000 000 do kolekcji.
  - Następnie pobrano 500 000 na podstawie klucza i dodano do drugiej kolekcji.
  - Rezultatem testu jest całkowity czas wykonania operacji.
  - W przypadku języka Java:
    - wykorzystano Integer (obiekt) zamiast int (wartość),
    - należało zwiększyć Java VM heap momory (parametr VM: -Xms512M –Xmx2048M).
  - Wyniki (czas operacji)
    - MS C# 3.0: 10,42 s
    - Java: 6,58 s

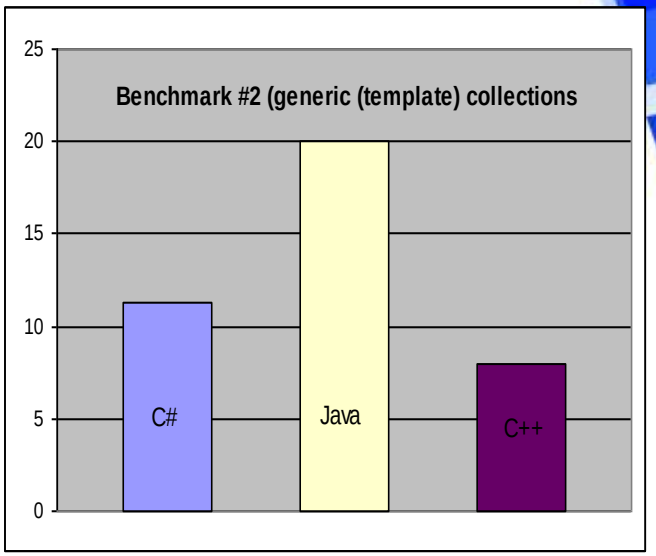
Testy wydajności (3)

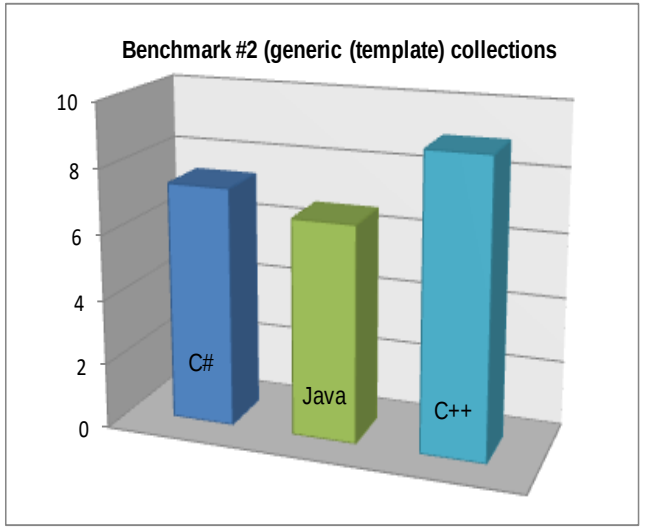
- Test #2
  - Podobnie jak test #1, ale wykorzystano klasy parametryzowane:
    - Dictionary dla C#,
    - generic HashMap dla Java,
    - STL std::map for C++.
  - Rezultatem testu jest całkowity czas wykonania operacji.
  - Uwagi:
    - W przypadku C++ wynik (czas) nie uwzględnia ręcznego zwolnienia pamięci (analogicznie jak dla C# i Java).
    - Dla języka Java należało zwiększyć Java VM heap.



Testy wydajności (4)

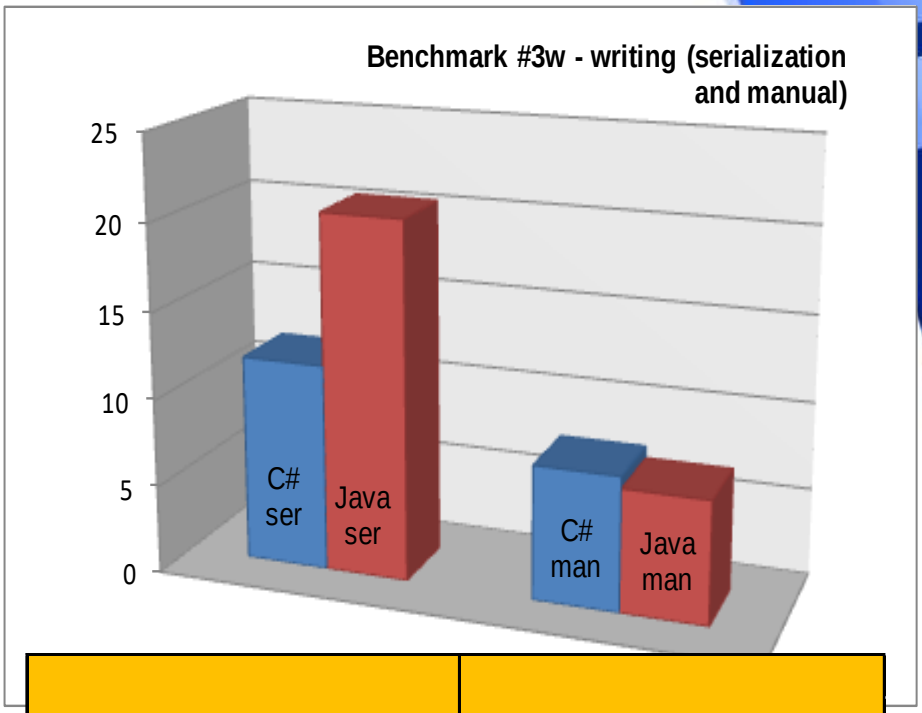
- Test #2 – c.d.
  - Zaskakujące zmiany w stosunku do starej wersji testów (Java 1.5, .NET 2.0, C++ 2005): zdecydowanie zmieniły się różnice wydajności pomiędzy C#, Java oraz C++ (mniej = lepiej). Uwaga: nie należy porównywać czasów bezwzględnych, a tylko różnice.
  - Wytłumaczenie?





Testy wydajności (5)

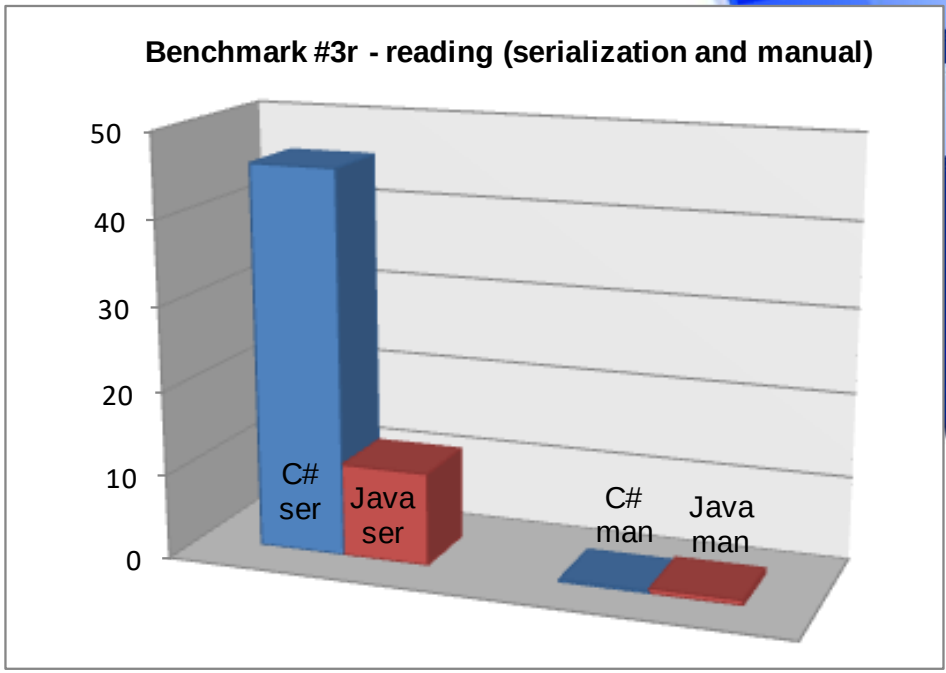
- Test #3w (write)
  - Test rozpoczyna się wykonaniem operacji z Test#2. Następnie zawartość drugiej kolekcji jest serializowana lub „ręcznie” zapisywana na dysku.



MS C# 3.0 (.NET 3.5SP1)		Java*	
[seconds] (less is better)			
Serialization	**Maunal**	**Serialization**	**Maunal**
11,86	7,68	20,52	7,01
File size: 28 390 349 bytes	File size: 14 388 893 bytes	File size: (21 389 128 bytes	File size: 14 961 597 bytes

Testy wydajności (6)

- Test #3r (read)
  - Odczytanie z dysku kolekcji zapisanej w czasie testu Test#3r.



MS C# 3.0 (.NET 3.5SP1)		Java*	
[seconds] (less is better)			
45,68	0,29	11,03	0,63

Podsumowanie

- Język Java zawiera wiele użytecznych konstrukcji.
- Wyjątki są eleganckim rozwiązaniem ułatwiającym odnalezienie błędu.
- Pojemniki umożliwiają łatwe zarządzanie zmienną liczbą elementów.
- Nowe wersje języka rozszerzają oraz ulepszają istniejące rozwiązania:
  - Java generics,
  - Enum,
  - Nowa pętla For
- Wydajność współczesnych języków programowania (Java, MS C#, MS C++) jest dość zbliżona.