Design and Analysis of Information Systems (MAS)

Mariusz Trzaska, Ph. D.
mtrzaska@pjwstk.edu.pl

Lecture 03

Some Constructs of Object-Oriented Programming Languages (2)

Continuation of the previous lecture

These slides make use of the following sources: Thinking in Java (3rd edition) by Bruce ʼa Eckel ʼa

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

Storing of many items

- A purpose
- Arrays
- Containers
  - Collections
  - Maps
- Pros and Cons

Arrays

- Pros
- Cons

```
int[] a1 = { 1, 2, 3, 4, 5 };
int[] a2 = a1; // what is going on here?
```

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

- Useful methods: *java.util.Arrays*

Collections

- The List: order
  - `ArrayList`: quick access to items with a particular index, quite slow in insertions

```
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`: very fast insertions, removing and quite good speed in accessing items

Collections (2)

- `Set`: no orders and repetitions
  - *HashSet*: fast searching
  - *TreeSet*
- Functionalities
  - `boolean add(Object)`
  - `boolean addAll(Collection)`
  - `void clear( )`
  - `boolean contains(Object)`
  - `boolean containsAll(Collection)`
  - `boolean isEmpty( )`

Collections (3)

- Functionalities – *cont.*
  - `Iterator iterator( )`
  - `boolean remove(Object)`
  - `boolean removeAll(Collection)`
  - `boolean retainAll(Collection)`
  - `int size( )`
  - `Object[]toArray( )`
  - `Object[]toArray(Object[] a)`

Maps

- Stores: a key → a value
- A `hash`ing function and `equals` method
- Different kinds
  - `HashMap` (based on the hashtable)
  - `TreeMap` (red-black trees)
- Pros and cons

Containers - performance

- List (least = better)

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

Containers - performance (2)

- Set (least = better)

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

Containers - performance (3)

- Map (least = better)

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

Collections – useful functionalities

- The **Collections** class:
  - `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*

- What exactly is stored in a container?
- What is wrong with the following code?

```
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)

- The solution: generic classes

```
private void testCollection() {
    List<String> list = new ArrayList<>();
    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);
    }
  }
```

A new *for* loop

- An old-school approach (not recommended)

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

- The new approach (recommended)

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

I/O System

- An input stream
  - An array of bytes
  - A String object
  - A file
  - A sequence of other streams
  - Others, i.e. a network connection
- An output stream
- Readers
- Writers

I/O System (2)

- Reading a file line by line

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

- Reading a data from the memory

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

I/O System (3)

- Writing to a file line by line

```
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");
}
```

I/O System (4)

- Writing and reading to/from a file

```
try {
      final String fileName = "Data.bin";

      DataOutputStream out2 = new DataOutputStream(new BufferedOutputStream(
            new FileOutputStream(fileName)));
       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(fileName)));
       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);
}
```

I/O System (5)

- Redirecting of the standard in/out streams
  - `System.setIn(InputStream)`
  - `System.setOut(PrintStream)`
  - `System.setErr(PrintStream)`

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

I/O System (6)

- An old and new APIs:
  - `java.io.*`
  - `java.nio.*` (performance)
- *Memory-mapped* files
- Locking files
- Compression
  - `ZipOutputStream`
  - `GZIPOutputStream`
  - `ZipInputStream`
  - `GZIPInputStream`
- JAR files
- Serialization

I/O System – the File Class

- `java.io.File`
  - Searching for files
  - Informations about files
    - Attributes,
    - Location.
  - Deleting files
  - Making directories
  - Deleting directories
  - Changing a file's name
  - `pathSeparator`

Remarks on GUI Implementation

- Adding objects, not `Strings` to widgets - overriding the `toString()` method. This means, that you do not have to search for objects based on the index.
- Wrapping business code into methods, rather than placing it directly in the event handling methods (listeners).

Remarks on GUI Implementation (2)

- Each control has its own listener (not sharing the listener without a good reason).
- Using "logical" events/listeners (e.g. `actionPerformed`, `selectionChanged`), not hardware ones (e.g. `MouseListener`).

Java JDK 7 – What's New?

- Published: 2011-07-28.
- Support for dynamically typed languages.
- The `string` type in the `switch` construct.
- A new construct `try-with-resources` and multi `catch`.
- Type inference – the diamond notation <>.
- Simplified method calls with many parameters.

Java JDK 7 – What's New? (2)

- Improved support for collections (including the multi-threaded ones).
- Unicode 6.0.
- A new API for I/O operations
  - Files;
  - Sockets.
- Improved Java2D.
- A new L&F: Nimbus.
- A new possibilities regarding sound synthesis.

Java JDK 7 – What's New? (3)

- Better XML support.
- Proposals for the future version JDK 8 (2014-03?):
  - Better annotations;
  - Improved collection management;
  - The Lambda project;
  - Modularization.

See: http://openjdk.java.net/projects/jdk7/features/

Java JDK 8 – What's New?

- Published: 2014-03.
- `Static` methods in interfaces.

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

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

Java JDK 8 – What's New? (2)

- Default implementations of methods in interfaces (not for the `Object` class). Also in the MS C# 8.0.

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

- Functional interfaces (contains a single method only). Useful for lambda expressions, method or constructor references.

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

Java JDK 8 – What's New? (3)

- Lambdas
  - Functionality as a method argument.
  - Their compatibility is defined using types of input, output and exceptions.

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

  - "Capturing" lambdas – access outside parameters/variables (only for "`finals`").

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

Java JDK 8 – What's New? (4)

- New functional operations on streams of values (e.g. collections): `java.util.stream`.
  - A stream can be processed only once;
  - Sequential or parallel (multiple threads);
  - A fluent API:

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

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

Java JDK 8 – What's New? (5)

- New functional operations on streams of values (e.g. collections) – *cont*.:
  - Intermediate operations ("lazy" approach):
    - `filter, map, flatMap, peek, distinct, sorted, limit, substream`
  - Terminal operations:
    - `forEach, toArray, reduce, collect, min, max, count, anyMatch, allMatch, noneMatch, findFirst, findAny`

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

Java JDK 8 – What's New? (6)

- New functional operations on streams of values (e.g. collections) – cont.:
  - Object and primitive streams:

```
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>
       ...
```

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

Java JDK 8 – What's New? (7)

- Method references.
- New functional interfaces in `java.util.function`.
- Generic type inference improvements.
- The new Date/Time class (finally!): the `java.time` package (immutable values)
  - `LocalDate`, `LocalDateTime`, `Period`,
  - easy instantiating,
  - easy manipulation.

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

Java JDK 8 – What's New? (8)

- New methods added to the Collections API (using the default interface implementation).
- Concurrency API improvements.
- Support for the new stream concept in IO/NIO functionality.

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

- Reflection and annotation changes.
- The new JavaScript engine (Nashorn).

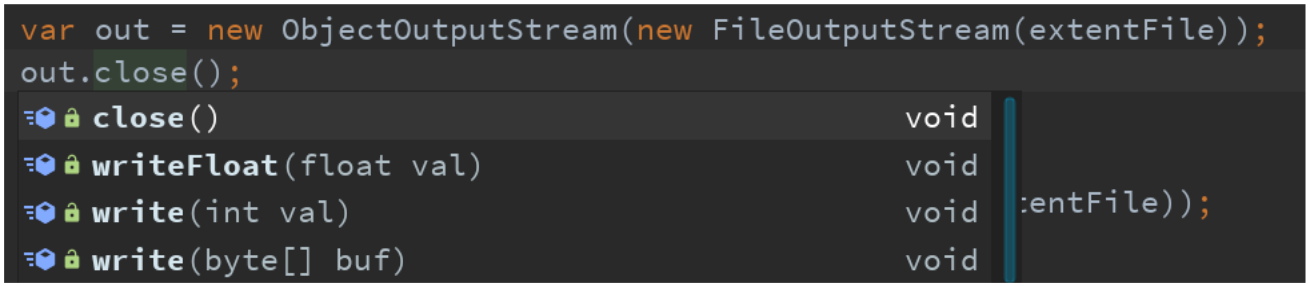
See: http://www.techempower.com/blog/2013/03/26/everything-about-java-8/, http://www.infoq.com/news/2013/08/everything-about-java-8

Java JDK 10 – What's New? (9)

- New keyword ***var*** (similarly like in MS C#):
  - Local variable type inference (strongly typed based on the right-side value)

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

  - suggestions from IDE



More info: *https://stackify.com/whats-new-in-java-10/*

Java JDK 11 – What's New? (10)

- Java FX improvements
- direct running of codes (scripts) that are in a single file with the source code, e.g. *java.exe ExampleCode.java*,
- additional methods to work with the `Strings`,
- `var` in lambda expressions,
- new garbage collectors.

More info: *https://javastart.pl/b/newsy/java-11-co-nowego/*

Java JDK 12 – What's New? (11)

- better `switch`

```
int numLetters = switch (day) {
        case MONDAY, FRIDAY, SUNDAY -> 6;
        case TUESDAY -> 7;
        case THURSDAY, SATURDAY -> 8;
        case WEDNESDAY -> 9;
};
```

- better garbage collectors,
- added *micro-benchmarks* to JDK.

More info: https://bulldogjob.pl/news/545-java-12-nowe-funkcje

Java JDK 13 – What's New? (12)

- text blocks """

```
"""
line 1
line 2
line 3
"""
```

- additional improvements in `switch`,
- rewritten socket API,
- better garbage collectors.

More info: https://bulldogjob.pl/news/778-java-13-co-nowego

Java JDK 14 – What's New? (13)

- better handling of `NullPointerException`

```
a.i = 7;

Exception in thread "main" java.lang.NullPointerException:
Cannot assign field "i" because "a" is null
at TestJEP358.main(TestJEP358.java:5)
```

- special classes for *immutable data*,

```
record Point(int x, int y) { }
```

- better garbage collectors.

More info: https://bartlomiejchmielewski.pl/java-14-co-nowego/

Java JDK 15 – What's New? (14)

- *Sealed Classes* – allows inheriting from a class only for selected classes (see also `final`)

```
public sealed class Shape
    permits Circle, Rectangle, Square {...}
```

- *Text Blocks*

```
String text = """
              line 1
              line 2
              line 3
""";
```

- ...

More info: https://bartlomiejchmielewski.pl/java-15-co-nowego/

Java JDK 16 – What's New? (15)

- A few different changes and the final versions of the test mechanisms:
  - *Pattern Matching for instanceof,*
  - *Records, a class containing*

```
public record Point(int x, int y) {}
```

    - attributes: `final x` and `y`,
    - the canonical constructor with all the attributes,
    - read-only methods: `x()` and `y()`,
    - methods: `equals()`, `hashCode()`, `toString()`,
    - a possibility of adding new methods, attributes, etc.

More info: https://bartlomiejchmielewski.pl/java-16-nowosci/

Java JDK 17 LTS – What's New? (16)

- Some less important changes and final version:
  - *Sealed Classes*,
- *Pattern Matching for switch* (Preview)

```
static String formatterPatternSwitch(Object o) {
    return switch (o) {
        case Integer i -> String.format("int %d", i);
        case Long l    -> String.format("long %d", l);
        case Double d  -> String.format("double %f", d);
        case String s  -> String.format("String %s", s);
        default        -> o.toString();
    };
}
```

More info: https://bartlomiejchmielewski.pl/java-17/

Java JDK 18 – What's New? (18)

- Default character set: UTF-8 (and not depending on the JRE version).
- *Simple Web Server* as the `jwebserver` command and the `SimpleFileServer` class.
- Improvements in the reflection mechanism.
- Announcement of the `Finalize()` method removal.
- A new JavaDoc tag `@snippet`.

More info: *https://bartlomiejchmielewski.pl/java-18/*

Java JDK 19 – What's New? (19)

- `System.out` uses the system character encoding by default;
- Easier creation of `HashMap` with the given capacity:

```
Map<String, Integer> map = HashMap.newHashMap(120);
```

- More improvements with patterns in the `switch` command;
- Improvements of patterns with `instanceof` and `record`;

More info: https://www.happycoders.eu/java/java-19-features/

Java JDK 19 (2) – What's New? (20)

- Virtual threads (preview): no direct 1:1 mapping to the system threads;
- Preview of the *Structured Concurrency*;
- Preview of the *Foreign Memory Access API* and *Foreign Linker API* as a replacement for the *Java Native Interface* (JNI): access to memory/methods outside of the VM (e.g. C libraries);
- Preview of the *Vector API* math calculations library.

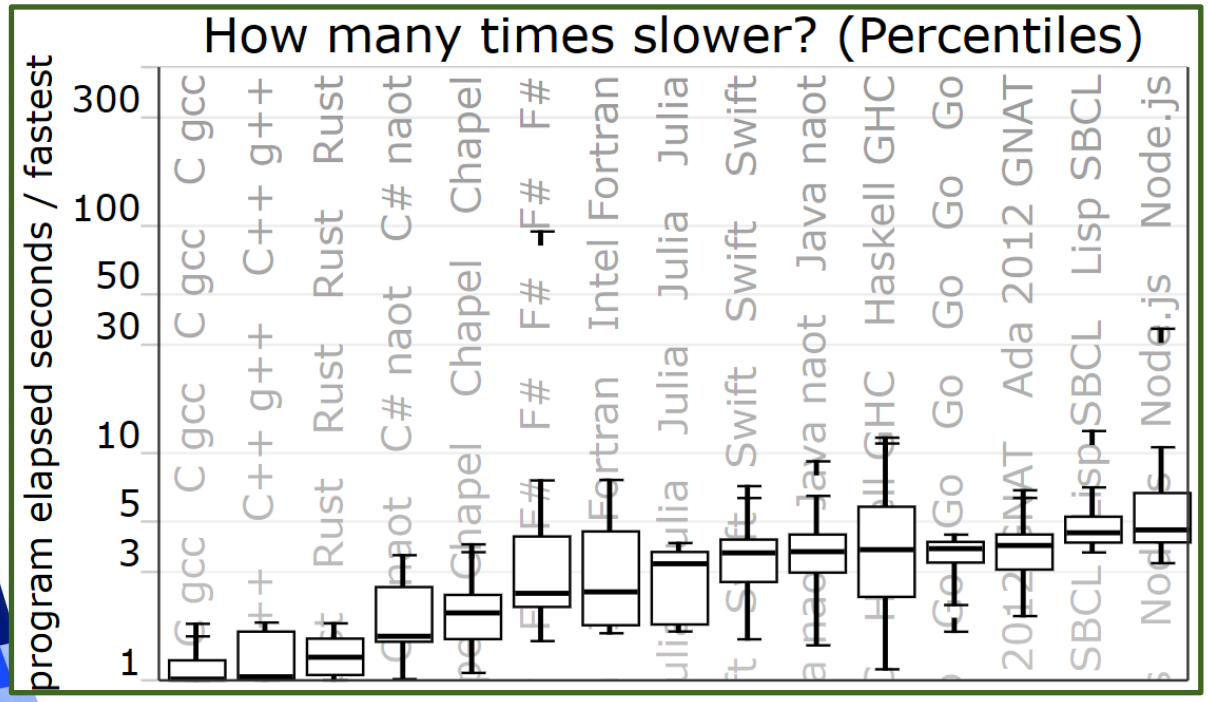
More info: https://www.happycoders.eu/java/java-19-features/

Programming Languages Performance

- Which of the programming languages is faster?
  - Java?
  - MS C#?
  - MS C++?
  - Rust?
- Is Java really much slower then C++?
- There are many different opinions…

Programming Languages Performance (2)

- Different approaches to benchmarking including speed/time, memory, CPU etc.
  - The Computer Language 25.02 Benchmarks Game (Debian)



Source: https://benchmarksgame-team.pages.debian.net/benchmarksgame/box-plot-summary-charts.html

Programming Languages Performance (3)

- Different approaches to benchmarking including speed/time, memory, CPU etc. – *cont.*
  - Programming Language and compiler Benchmarks (Vercel)
  - https://github.com/kostya/benchmarks

Summary

- The Java language contains many useful structures.
- The exceptions are an elegant solution simplifying finding an error.
- Containers allow easy storing and maintain many items.
- New versions of the Java language improves and extends existing solutions:
  - Java generics,
  - Enum,
  - The new For loop.
- The performance of the nowadays programming languages (Java, MS C#, MS C++) is quite similar.