Design and Analysis of Information Systems (MAS)

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

Lecture 04

Classes in Object-Oriented Programming Languages

Outline

- Implementation of the following constructs in the popular programming languages (Java):
  - A class
  - An extent
  - Attributes:
    - Simple and complex
    - Mandatory and optional
    - Single and multi-valued
    - Of an object or a class
    - Derived
  - Methods:
    - of a class
    - Of an object
    - Overloading and overriding
  - Persistency
  - Utility class
- Summary

The Class

- A class is a named description of a group of objects which share the same set of invariants (properties).
- The class is not a set of objects,
- **The class describes an object**.

PRI lectures by Ewa Stemposz, Ph.D. and prof. Kazimierz Subieta.

Attributes

- They are utilized to describing properties of objects.
- Kinds of:
  - Simple and complex,
  - Mandatory and optional,
  - Single and multi-valued,
  - Of an object or a class,
  - Derived.

Methods

- Allowing performing of operations on objects which usually lead to changing of their states.
- Kinds of
  - **An object method**. Performs an operation (and has access to) on one, particular object. The one, for which has been called on.
  - **A class method**. Has access to the entire extent of the class. Called on a class (rather then an object). Such an approach allows calling them even there are no existing objects of the class.

Classes and Programming Languages

- How the definition is related to popular programming languages?
- In languages:
  - Java,
  - MS C#,
  - C++

  classes exist according to the definition.
- Unfortunately it is not true in case of every construct (term, definition) known from object-orientedness (UML).

Classes in Java

- Let's assume that we need a class describing a movie in a video store.

```
**
 * Information about a movie.
 *
*/
public class Movie {
        /* Class body */
}
```

Class Extent

- A set of currently existing objects of the class.
- In languages
  - Java,
  - MS C#,
  - C++

  class extent **does not exist**.
- What we can do about it?
- Our own implementation of the extent.

Class Extent Implementation

- Two different approaches. Implementation:
  - In the same business class
  - Using an additional class
    - Business class Movie, extent i.e. ***Movies***
    - Business class Movie, extent i.e. ***MovieExtent***
- Which approach is better?
  - Pros
  - Cons
- What about using a database?

Class Extent Implementation (2)

- Implementation in the same business class
  - A container storing references to objects of the class (declared as a class attribute - `static`),
  - Supporting methods
    - Add,
    - Remove,
    - Search.
    - ?
  - Implementation of the class methods. In the same class with the `static` keyword.

Implementation in the same Business Class

```java
public class Movie {
   public Movie() {
      // Add to the extent
      addMovie(this);
   }

   /**
    * The extent. Non-final required - see further (persistency).
   */
   private static List<Movie> extent = new ArrayList<>();

   /**
    * Adds a movie to the extent.
    *
    * @param movie the movie
    */
   private static void addMovie(Movie movie) {
      extent.add(movie);
   }

   /**
    * Removes a movie from the extent.
    *
    * @param movie the movie
    */
   private static void removeMovie(Movie movie) {
      extent.remove(movie);
   }

   // [...]
}
```

Implementation in the same Business Class (2)

```
public class Movie {
   // [...]

   /** Shows the extent (utility class method). */
   public static void showExtent() {

      System.out.println("Extent of the class: " + Movie.class.getName());

      for (Movie movie : extent) {
         System.out.println(movie);
      }
   }
}
```

```
   private static void test1() {

      // A test: Class extent implemented in the same class
      Movie movie1 = new Movie("Terminator 1", LocalDate.now(), 29.90f);
      Movie movie2 = new Movie("Terminator 2", LocalDate.now(), 34.90f);

      Movie.showExtent();
   }
```

```
Extent of the class: mt.mas.Movie
Movie: Terminator 1, id: mt.mas.Movie@23e028a9
Movie: Terminator 2, id: mt.mas.Movie@63e2203c
```

Class Extent Implementation (3)

- Using an external class
  - Names
    - Business class Movie, extent i.e. ***Movies***
    - Business class Movie, extent i.e. ***MovieExtent***
  - Possibility of creating many different extents (why?).
  - A container storing references to objects of the business class,
  - Utility methods
    - Add,
    - Remove,
    - Search,
    - …

Class Extent Implementation (4)

- Using an additional class – cont.
  - Implementation of the class methods. They will be outside of the business class – in the class which manages the extent.
  - Possible problems with access to hidden attributes/methods (`public/private/protected`).
  - Automatic adding to the extent could be more difficult.
  - Other possibilities, e.g.
    - Inner class,
    - References collection with the `static` keyword.

Class Extent Implementation Using an Additional Class

```java
public class Movie {
    // ... Class body (business related)
}
```

```java
public class MovieExtent {
    /** The extent. */
    private List<Movie> extent = new ArrayList<>();

    public void addMovie(Movie movie) {
        extent.add(movie);
    }

    public void removeMovie(Movie movie) {
        extent.remove(movie);
    }

    public void showExtent() {
        System.out.println("Extent of the class: " + Movie.class.getName());
        for (Movie movie : extent) {
            System.out.println(movie);
        }
    }
}
```

Class Extent Implementation Using an Additional Class (2)

```
private static void testExternalExtent() {
   // A test: Class extent implemented using an external class
   MovieExtent movieExtent = new MovieExtent();

   Movie movie1 = new Movie();
   movieExtent.addMovie(movie1);

   Movie movie2 = new Movie();
   movieExtent.addMovie(movie2);

   movieExtent.showExtent();
}
```

```
Extent of the class: mt.mas.Movie
Movie: Terminator 1, id: mt.mas.Movie@3dd4520b
Movie: Terminator 2, id: mt.mas.Movie@1efed156
```

Attributes in Object-orientedness and Java

- Kinds of
  - **Simple**. They appear in the same form like in the UML.

```
public class Movie {
    private float price;
}
```

  - **Complex**. A complex attribute is described using another class (i.e. date). Consequences:
    - In a business class (i.e. movie) we store a reference rather then a value.
    - That means that we can share it. This is different then in the theoretical semantic of a complex attribute., i.e. another object can have a reference to the same date.

```
public class Movie {
    private LocalDate additionDate;
}
```

    - When we should use **an attribute** and when **an association with a class**?

Attributes in Object-orientedness and Java (2)

- Kinds of – cont.
  - **Mandatory**.
    - Every simple attribute stores some value – there is no way to not storing anything.
    - **A complex attribute** stores a reference for an object being his value. Because it is a reference it could be `null` which means „no value”.
    - We should check if the proper value is provided.
    - Possibly annotation: `@NotNull`.

Attributes in Object-orientedness and Java (2)

- Kinds of – cont.
  - **Optional**
    - A proper **storing** and **processing** data.
    - For complex attributes we can store a `null` as an information about lack of value.
    - How about simple attributes? Wrapper classes!
    - Make sure that you are ready for the lack of value.
    - Possibly annotation: `@Nullable`.

Attributes in Object-orientedness and Java (2)

- **Optional attributes** – *cont.*
  - Another possibility is to use `Optional` (Java 8+), but there are some problems, e.g. serialization.

```
public class Employee {

    // ...

    private Optional<Double> extraBonus = Optional.empty(); // initialization without a value

    public Optional<Double> getExtraBonus() {
        return extraBonus;
    }

    public void setExtraBonus(Optional<Double> extraBonus) {
        this.extraBonus = extraBonus;
    }

    public double getIncome() {
        return getSalary() + getExtraBonus().orElse(0d);
    }

    @Override
    public String toString() {
        return String.format("Emp '%s', sal: %s, bonus: %s", getName(), getSalary(),
             getExtraBonus().isPresent() ? getExtraBonus().get() : "(no bonus)");
    }
}
```

Attributes in Object-orientedness and Java (3)

- Kinds of – cont.
  - **Single**. The same story like in the UML.
  - **Multi-valued**. Use array or containers (preferred solution).
  - **Object attributes**. The same story like in the UML.
  - **Class attributes**. Implementation depends on the way of dealing with the class extent:
    - An extent in the same business-class ➔ class attributes in the same class with the `static` keyword,
    - An extent in additional class ➔ class attributes in the additional class (without the `static` keyword).

Attributes in Object-orientedness and Java (4)

- Kinds of – cont.
  - Derived.
    - In case of:
      - Orthodox encapsulation, special treatment of an attribute could be implemented in getter (`getXXX`) methods. In special and rare circumstances also in setters (`setXXX`). In most cases, there is no „real" attribute.
      - Direct access to an attribute, there is no way to control the behaviour.
    - A perfect construct in C#: *properties*.

```
private float price {
    get { return price * taxFactor; }
}
```

Methods in Object-orientation and Java

- Kinds of:
  - **An object method**. The same semantics like in the UML.

```
public float getPrice() {
     return price;
}
```

  - **A class method**. Particular implementation depends on the way of dealing with the class extend (i.e. `void showExtent()`):
    - An extent in the same business class ➔ class methods are in the same class with the `static` keyword,
    - An extent as an additional class ➔ class methods in the additional class (without the `static` keyword).

Methods in Object-orientation and Java (2)

- **An overloading of a method**. The same semantics like in the UML.

```
public float getPrice() {
    return price;
}
public float getPrice(float vatRate) {
    return price * (1.0f + vatRate / 100.0f);
}
```

- **An overriding of a method**. The same semantics like in the UML.

```
public class Movie{
    // [...]

    @Override
    public String toString() {
        return „Movie: " + title;
    }}
```

The extent of the Movie class:

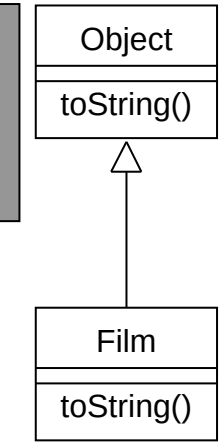
mt.mas.Film@126804e

4c3

The extent of the Movie class:

Movie: Terminator 1

Movie: Terminator 2



A Persistency of an Extent

- A class extent is persistent if the objects will „survive" shutdown of the system – after rebooting there will be the same objects.
- In languages
  - Java,
  - MS C#,
  - C++

  the property does not exist directly.
- Hence we can implement it manually writing to and reading from a file.

A Persistency of an Extent (2)

- As a result we have similar objects rather then the same objects.
- An implementation
  - Manual
    - Speed of working,
    - Full control of the result,
    - A better resistance to source code changes,
    - A smaller file,
    - Requires (usually) a lot of work.

A Persistency of an Extent (3)

- An implementation – *cont*.
  - Using the serialization technique,
    - Easy to use,
    - Usually slower,
    - A bigger file,
    - Compatibility/migration issues,
    - A possibility for modifications:
      - By adding methods:
        - `private void writeObject(ObjectOutputStream stream) throws IOException`
        - `private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException`
      - Using the `transient` keyword.

A Persistency of an Extent (4)

- An implementation – *cont.*
  - Using a database,
    - A necessity of mapping Java structures onto database structures and vice versa,
    - Possibility of using a query language (i.e. SQL),
    - Possibility of working with different database systems,
    - A bigger file,
    - A better speed of working.
  - Using an additional library (e.g. ORMs).
    - Hibernate (*http://www.hibernate.org/*)
    - Java Persistence API (*https://glassfish.dev.java.net/*)
    - Java Data Objects (*http://www.jpox.org/*)
    - …

A Persistency of an Extent – a manual implementation

- There are reading/writing methods in each business class.

```
public class Movie {
   private String title;
   private float price;
   private LocalDate additionDate;  // requires Java 8+

   private void write(DataOutputStream stream) throws IOException {
      stream.writeUTF(title);
      stream.writeFloat(price);
      stream.writeLong(additionDate.toEpochDay());    // count of days where day 0 is 1970-
01-01 (ISO)
   }

   private void read(DataInputStream stream) throws IOException {
      title = stream.readUTF();
      price = stream.readFloat();
      long epochDay = stream.readLong();
      additionDate = LocalDate.ofEpochDay(epochDay);
   }

   // [...]
}
```

A Persistency of an Extent – a manual implementation (2)

- In every class, which manages an extent there are methods reading and writing the entire extent.

```
public class Movie {
   //  [...]

   private static List<Movie> extent = new ArrayList<>();

   public static void writeExtent(DataOutputStream stream) throws
IOException {
      // Number of objects
      stream.writeInt(extent.size());

      for (Movie movie : extent) {
         movie.write(stream);
      }
   }

   public static void readExtent(DataInputStream stream) throws
IOException {
      Movie movie = null;

      // Get the number of written objects
      int objectCount = stream.readInt();

      // remove the current extent
      extent.clear();

      for (int i = 0; i < objectCount; i++) {
         movie = new Movie();
         movie.read(stream);
      }
   }
}
```

A Persistency of an Extent – a manual implementation (3)

```
private static void testExtentManual() {
   final String extentFile = "d:\\temp\\mas-extent.bin";

   // A test: persistency of the extent (manual impl.)
   try {
      // Write the extent to the given stream
      DataOutputStream out2 = new DataOutputStream(new BufferedOutputStream(new
                                                FileOutputStream(extentFile)));
      Movie.writeExtent(out2);
      out2.close();

      // Read the extent from the given stream
      DataInputStream in2 = new DataInputStream(new BufferedInputStream(new
                                                FileInputStream(extentFile)));
      Movie.readExtent(in2);
      in2.close();
   } catch (IOException e) {
      e.printStackTrace();
   }

   Movie.showExtent();
}
```

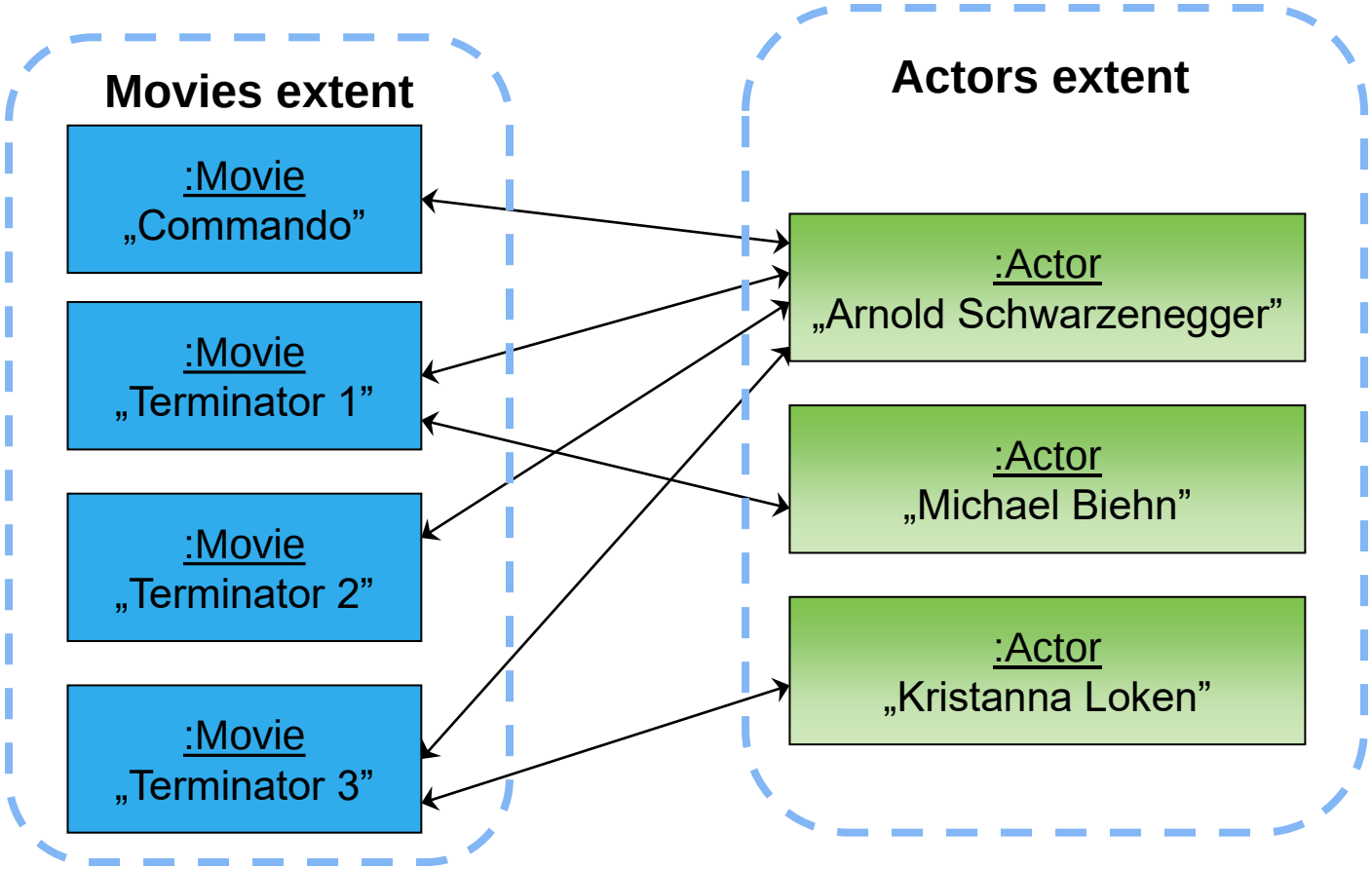
A Persistency of an Extent – a manual implementation (4)

- The example shows writing and reading of the extent.
- Thanks to the splitting of the functionality into a couple of methods, it is possible to store many extents in the same stream
- As the result state of the entire application is remembered in a single file.
- The sample result
  - An extent created in the memory,
  - An extent written and read from a file.

```
Extent of the class: mt.mas.Movie
Movie: Terminator 1, id: mt.mas.Movie@23e028a9
Movie: Terminator 2, id: mt.mas.Movie@63e2203c
Extent of the class: mt.mas.Movie
Movie: Terminator 1, id: mt.mas.Movie@6737fd8f
Movie: Terminator 2, id: mt.mas.Movie@72b6cbcc
```

A Persistency of an Extent – links problem

- The presented approach to extent persistency is very simple. It does not treat linked objects in a right way.



A Persistency of an Extent – a serialization implementation

- The serialization is a mechanism implemented in one of the Java libraries.
- It allows automatically:
  - Writing a graph of objects to a stream,
  - Reading the graph of objects from a stream.
- The only requirement to utilize the serialization is special implementation of the **`Serializable`** interface by the serialized classes (and by their content).
- The special implementation of the interface means that we only need to declare the implementation but without providing necessary methods' bodies. The bodies are implemented by the „library"

A Persistency of an Extent – a serialization implementation (2)

- Declare implementation of the interface by a business class.

```
public class Movie implements Serializable {
    //  [...]
    private String title;
    private float price;
    private LocalDate additionDate;  // requires Java 8+
}
```

- Create methods for writing and reading the extent.

```
public class Movie implements Serializable {
    //  [...]
    // Non-final required because of (de)serialization (persistency).
    private static List<Movie> extent = new ArrayList<>();

    public static void writeExtent(ObjectOutputStream stream) throws IOException {
        stream.writeObject(extent);
    }

    public static void readExtent(ObjectInputStream stream) throws IOException,
ClassNotFoundException {
        extent = (ArrayList<Movie>) stream.readObject();
    }
}
```

- All the extents (of all classes) **have to be written to the same stream** (file).

A Persistency of an Extent – a serialization implementation (3)

- An utilization

```
try {
    // Write the extent to the given stream
    var out = new ObjectOutputStream(new FileOutputStream(extentFile));
    Movie.writeExtent(out);
    // Write any other extents
    out.close();

    // Read the extent from the given stream
    var in = new ObjectInputStream(new FileInputStream(extentFile));
    Movie.readExtent(in);
    // Read any other extents
    in.close();
} catch (IOException | ClassNotFoundException e) { e.printStackTrace(); }
```

- Size of the extent file:
  - Manual implementation: 56 bytes,
  - Serialization: 263 bytes.
  - For bigger files the differences are less – about x2.

```
Extent of the class: mt.mas.Movie
Movie: Terminator 1, id: mt.mas.Movie@61a485d2
Movie: Terminator 2, id: mt.mas.Movie@1810399e
Extent of the class: mt.mas.Movie
Movie: Terminator 1, id: mt.mas.Movie@2f686d1f
Movie: Terminator 2, id: mt.mas.Movie@3fee9989
```

An Extent Management

- Presented approaches to an extent implementation will be (almost) the same for each business class in a system.
- Is there a way to unified them. To avoid writing many times the same code?
- Of course – we will utilize an inheritance in the Java language.
- Another approach: template classes (Java generics) – **a homework** for volunteers.

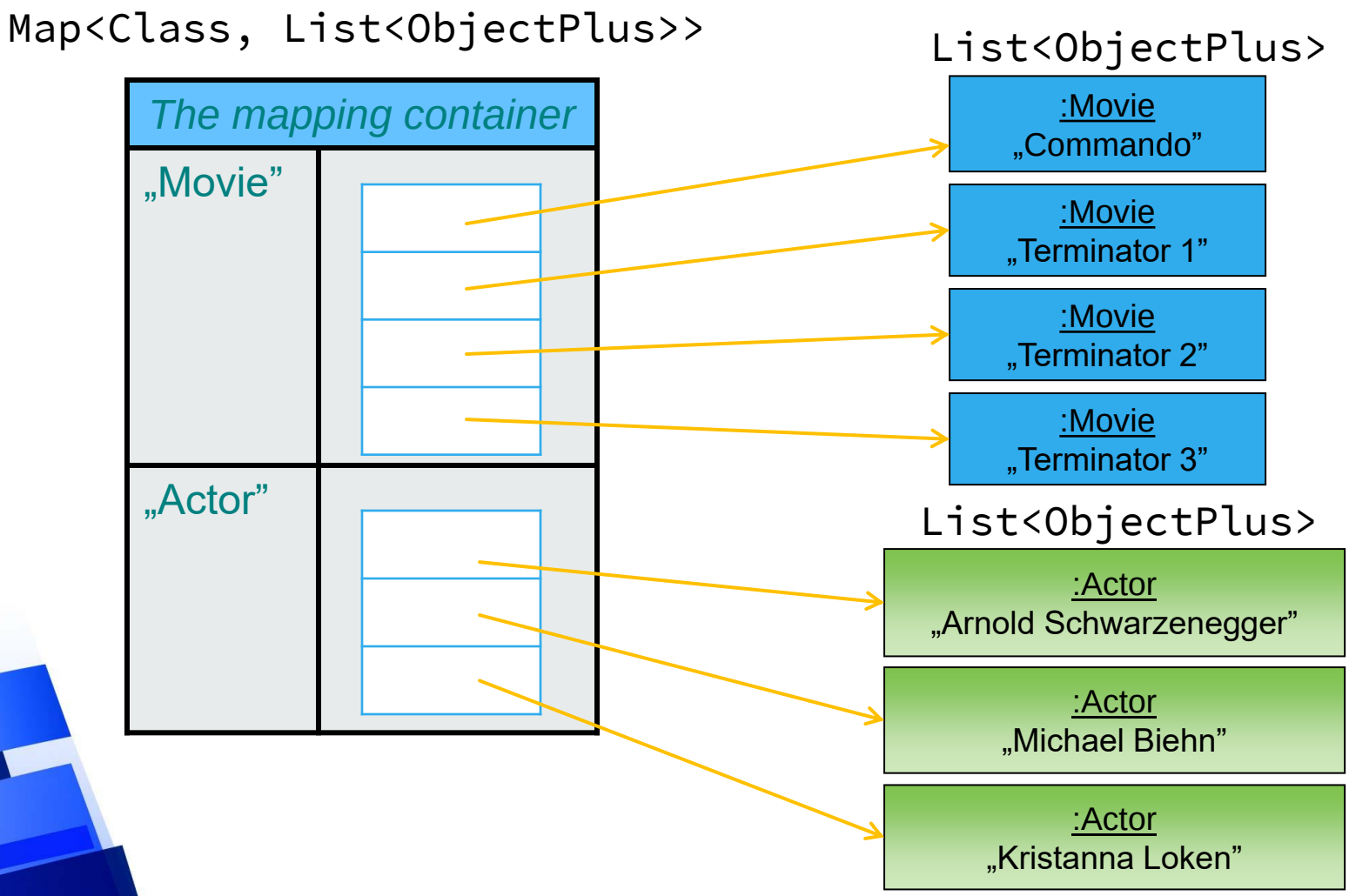
The Universal Extent Management

- Let's create a class which will be a super class for all business classes in our application.
- Let's call them the `ObjectPlus` and equip with:
  - A persistency,
  - An extent management,
  - ?
- We will use the first approach: a class extent within the same business class.

The Universal Extent

- Because all business classes inherit from the one class, we are not able to use an ordinary container.
- We will employ a special map storing keys and values:
  - The key will be a name of a particular business class,
  - The value will be a container storing its extent (references to all existing objects of the class).
- The map will contain not a single extent but all of them.

The Universal Extent (2)



The Universal Extent (3)

- The constructor of each business class will refer to a super constructor.

```
public class Movie2 extends ObjectPlus implements Serializable {

   //  Business implementation

   private String title;
   private float price;
   private LocalDate additionDate;

   /**
    * The constructor.
    */
   public Movie2(String title, LocalDate additionDate, float price) {
      // Call the constructor from the super class
      super();

      this.title = title;
      this.additionDate = additionDate;
      this.price = price;
   }

   // [...] business implementation
}
```

The Universal Extent (4)

- The superclass constructor takes care of adding the object to a proper extent.

```
public abstract class ObjectPlus implements Serializable {
   private static Map<Class, List<ObjectPlus>> allExtents = new Hashtable<>();

   /**
    * Constructor.
    */
   public ObjectPlus() {
      List<ObjectPlus> extent = null;
      Class theClass = this.getClass();

      if(allExtents.containsKey(theClass)) {
         // An extent of this class already exist
         extent = allExtents.get(theClass);
      }
      else {
         // An extent does not exist - create a new one
         extent = new ArrayList();
         allExtents.put(theClass, extent);
      }

      extent.add(this);
   }

   // [...]
}
```

The Universal Extent (5)

- It is also quite easy making the extents persistent (using the serialization).

```
public class ObjectPlus implements Serializable {
   private static Map<Class, List<ObjectPlus>> allExtents = new Hashtable<>();

   // [...]

   public static void writeExtents(ObjectOutputStream stream) throws IOException {
      stream.writeObject(allExtents);
   }

   public static void readExtents(ObjectInputStream stream) throws IOException,
                                 ClassNotFoundException {
      allExtents = (Hashtable) stream.readObject();
   }

   // [...]
}
```

Getting the extent

- We use a generic method
- Return `Iterable<T>` type

```
public class ObjectPlus implements Serializable {
   private static Map<Class, List<ObjectPlus>> allExtents = new Hashtable<>();

   // [...]

   public static <T> Iterable<T> getExtent(Class<T> type) throws
ClassNotFoundException {
      if(allExtents.containsKey(type)) {
         return (Iterable<T>) allExtents.get(type);
      }

      throw new ClassNotFoundException(
                     String.format("%s. Stored extents: %s",
                                    type.toString(),
                                    allExtents.keySet()));
   }}
```

- The result is of a specific type, e.g. `Iterable<Movie2>`

```
Iterable<Movie2> movieExtent =
ObjectPlus.getExtent(Movie2.class);
for (var movie : movieExtent) {
   System.out.println(movie.getTitle());
}
```

The Universal Class Methods

- Some of class methods could also benefit from the general functionality placed in the super class, i.e. showing the extent.

```
public class ObjectPlus implements Serializable {
   private static Map<Class, List<ObjectPlus>> allExtents = new Hashtable<>();

   // [...]

   public static void showExtent(Class theClass) throws Exception {
      List<ObjectPlus> extent = null;

      if(allExtents.containsKey(theClass)) {
         // Extent of this class already exist
         extent = allExtents.get(theClass);
      }
      else {
         throw new Exception("Unknown class " + theClass);
      }

      System.out.println("Extent of the class: " + theClass.getSimpleName());

      for(Object obj : extent) {
         System.out.println(obj);
      }
   }
}
```

The Universal Class Methods (2)

- Showing the extent using previously defined functionality.

```
public class Movie2 extends ObjectPlus implements Serializable {

   //  [...]

   public static void showExtent() throws Exception {
      ObjectPlus.showExtent(Movie2.class);
   }

   //  [...]
}
```

```
Extent of the class: Movie2
Movie: Terminator 1, id: mt.mas.Movie2@7a5d012c
Movie: Terminator 2, id: mt.mas.Movie2@68837a77
Extent of the class: Movie2
Movie: Terminator 1, id: mt.mas.Movie2@4c70fda8
Movie: Terminator 2, id: mt.mas.Movie2@224edc67
```

ObjectPlus v2?

- What can be improved in the proposed solution (`ObjectPlus`) in the area of the presented topics?

  - **A homework** for volunteers (description, source codes) with extra points to the exam.

The Summary

- Some terms belonging to object-orientedness do exist in popular programming languages.
- Unfortunately, some of them do not exist with the same semantics.
- In most cases, the non existing constructs could be implemented:
  - manually using a few different approaches,
  - Using some existing libraries.
- It is possible and useful to put the entire functionality for managing the class extent in a special super class.

Source files

Download source files for all MAS lectures

http://www.mtrzaska.com/plik/mas/mas-source-files-lectures