POLSKO-JAPOŃSKA AKADEMIA TECHNIK KOMPUTEROWYCH

Modelowanie i Analiza Systemów informacyjnych (MAS)

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

Wykład 4

Wykorzystanie klas w obiektowych językach programowania

Zagadnienia

- Realizacja w popularnych językach programowania (Java) poniższych konstrukcji:
  - Klasa
  - Ekstensja
  - Atrybuty:
    - proste i złożone
    - wymagane i opcjonalne
    - pojedyncze i powtarzalne
    - obiektu i klasowe
    - Pochodne (wyliczalne)
  - Metody:
    - obiektu
    - klasowe
    - przesłonięcie i przeciążenie
  - Trwałość ekstensji
  - Klasa dedykowana
- Podsumowanie

Klasa

- Klasa jest nazwanym opisem grupy obiektów, które współdzielą ten sam zbiór własności (inwariantów).
- Klasa nie jest zbiorem obiektów,
- Stosunek klasa/podklasa oznacza, że obiekty podklasy posiadają wszystkie inwarianty nadklasy, plus (ewentualnie) inwarianty swoje. Np. klasa Student ma wszystkie inwarianty klasy Osoba, plus inwarianty własne.
- Klasa opisuje obiekt.

Wykłady z PRI autorstwa dr inż. Ewy Stemposz oraz prof. Kazimierza Subiety.

Atrybuty

- Służą do opisu własności obiektów.
- Rodzaje
  - Proste i złożone,
  - Wymagane i opcjonalne,
  - Pojedyncze i powtarzalne,
  - Obiektu i klasowe,
  - Pochodne (wyliczalne).

Metody

- Umożliwiają wykonywanie operacji na obiektach, prowadzących (zwykle) do zmiany ich stanu.
- Rodzaje
  - Metoda obiektu. Wykonuje operacje (oraz ma dostęp) na jednym, konkretnym obiekcie. Na tym, na rzecz, którego została wywołana.
  - Metoda klasowa. Ma dostęp do całej ekstensji klasy. Wywoływana jest na rzecz klasy, a nie obiektu. Dzięki temu można jej użyć nawet nie mając żadnego obiektu danej klasy.

Klasy, a języki programowania

- Jak ma się podana definicja do popularnych, obiektowych języków programowania?
- W językach
  - Java,
  - MS C#,
  - C++

  klasy występują w sposób zgodny z przytoczoną definicją.
- Niestety nie dotyczy to wszystkich pojęć znanych z obiektowości (UML).

Klasy w języku Java

- Załóżmy, że potrzebna nam jest klasa opisująca film w wypożyczalni wideo.

```
**
 * Movie information.
 *
*/
public class Movie {
        /* Class body */
}
```

Ekstensja klasy

- Zbiór aktualnie istniejących obiektów danej klasy.
- W językach
  - Java,
  - MS C#,
  - C++

  ekstensja klasy nie występuje.
- Co w takim razie możemy zrobić?
- Własna implementacja ekstensji klasy.

Implementacja ekstensji klasy

- Dwa różne podejścia. Implementacja:
  - W ramach tej samej klasy biznesowej
  - Przy użyciu klasy dodatkowej
    - Klasa `Film`, jej ekstensja np. `Filmy`
    - Klasa `Film`, jej ekstensja np. `FilmEkstensja`
- Które podejście jest lepsze?
  - Wady
  - Zalety
- A co gdy korzystamy z bazy danych?

Implementacja ekstensji klasy (2)

- Implementacja w ramach tej samej klasy
  - Kontener przechowujący referencje do obiektów danej klasy (jako atrybut klasowy czyli *static*),
  - Metody pomocnicze
    - Dodanie,
    - Usunięcie,
    - Wyszukanie.
    - ?
  - Realizacja metod klasowych. W tej samej klasie, ze słowem kluczowym `static`.

Implementacja w ramach tej samej klasy

```
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);
    }

    // [...]
}
```

Implementacja w ramach tej samej klasy (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
```

Implementacja ekstensji klasy (3)

- Implementacja przy użyciu klasy dodatkowej
  - Nazewnictwo
    - Klasa Film, jej ekstensja np. *Filmy*
    - Klasa Film, jej ekstensja np. *FilmEkstensja*
  - Możliwość stworzenia wielu różnych ekstensji
  - Kolekcja przechowująca referencje do obiektów klasy biznesowej,
  - Metody pomocnicze
    - Dodanie,
    - Usunięcie,
    - Wyszukanie.

Implementacja ekstensji klasy (4)

- Implementacja przy użyciu klasy dodatkowej – *c. d.*
  - Realizacja metod klasowych. Będą na zewnątrz klasy biznesowej – w klasie zarządzającej ekstensją.
  - Problemy z dostępem do atrybutów i metod (*public/private/protected*).
  - Utrudnione automatyczne dodawanie do ekstensji.
  - Inne rozwiązania, np.
    - z klasą wewnętrzną
    - kolekcją referencji jako atrybutem static

Implementacja ekstensji przy użyciu klasy dodatkowej

```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);
        }
    }
}
```

Implementacja ekstensji przy użyciu klasy dodatkowej (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
```

Atrybuty w obiektowości, a w języku Java

- Rodzaje
  - Proste. Występują w takiej postaci jak w obiektowości (w UML).

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

  - Złożone. Atrybut złożony jest opisywany za pomocą nowej klasy (np. data). Konsekwencje:
    - W klasie biznesowej (np. Film) przechowujemy referencję do jego wystąpienia, a nie (złożoną) wartość.
    - W związku z powyższym możemy go współdzielić (inaczej niż w „teoretycznej" semantyce atrybutu złożonego), np. inny obiekt może przechowywać referencję do tej samej daty.

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

    - Kiedy stosujemy **atrybut**, a kiedy **asocjację i klasę**?

Atrybuty w obiektowości, a w języku Java (2)

- Rodzaje – c. d.
  - Wymagane.
    - Każdy atrybut prosty przechowuje jakąś wartość – nie może nie przechowywać.
    - Atrybut złożony przechowuje referencję do obiektu „będącego jego wartością”. Ponieważ jest to referencja, może mieć wartość `null`, czyli „brak wartości”. Należy „ręcznie” sprawdzać czy jest różna od `null`.
    - Ewentualnie adnotacja `@NotNull`.

Atrybuty w obiektowości, a w języku Java (3)

- Rodzaje – c. d.
  - Opcjonalne
    - Właściwe zapamiętanie informacji lub jej braku.
    - Odpowiednie przetwarzanie **obu przypadków**.
    - Dla atrybutów złożonych przechowujemy `null` jako informację o braku wartości.
    - Co z atrybutami prostymi? Klasy opakowujące!
    - Warto również zadbać o specjalny dostęp do takiego atrybutu (bo może nie mieć wartości!).
    - Ewentualnie adnotacja `@Nullable`.

Atrybuty w obiektowości, a w języku Java (3)

- Atrybuty opcjonalne – c. d.
  - Ewentualnie korzystamy z `Optional` (Java 8+), ale uwaga na problemy z np. serializacją.

```
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)");
    }
}
```

Atrybuty w obiektowości, a w języku Java (4)

- Rodzaje – c. d.
  - Pojedyncze. Taka sama semantyka jak w obiektowości (w UML).
  - Powtarzalne. Należy wykorzystać tablice lub kontenery (rozwiązanie preferowane gdy liczba wartości jest zmienna).
  - Obiektu. Taka sama semantyka jak w obiektowości (w UML).
  - Klasowe. Sposób realizacji zależy od podejścia do ekstensji:
    - Ekstensja w ramach tej samej klasy ➔ atrybuty klasowe w tej samej klasie ze słowem kluczowym `static`,
    - Ekstensja jako klasa dodatkowa ➔ atrybuty klasowe w klasie dodatkowej (bez słowa kluczowego `static`).

Atrybuty w obiektowości, a w języku Java (5)

- Rodzaje – c. d.
  - Pochodne (wyliczalne).
    - W przypadku:
      - hermetyzacji ortodoksyjnej, specjalne traktowanie atrybutu zaimplementowane w metodzie udostępniającej jego wartość (*getXXX*). W rzadkich i szczególnych sytuacjach również modyfikacja (*setXXX*). Przeważnie brak „prawdziwego” atrybutu.
      - bezpośredniego dostępu do atrybutu, implementacja jego specjalnego traktowania jest mocno utrudniona (niemożliwa?).
    - Doskonały mechanizm: *properties* jak na razie tylko w języku C#.

```
private float cena {
     get { return cena_netto * wspPodatek; }
}
```

Metody w obiektowości, a w języku Java

- Rodzaje
  - Metoda obiektu. Taka sama semantyka jak w obiektowości (w UML).

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

  - Metoda klasowa. Sposób realizacji zależy od podejścia do ekstensji (np. `void pokazEkstensje()`):
    - Ekstensja w ramach tej samej klasy ➔ metody klasowe w tej samej klasie ze słowem kluczowym `static`,
    - Ekstensja jako klasa dodatkowa ➔ metody klasowe w klasie dodatkowej (bez słowa kluczowego `static`).

Metody w obiektowości, a w języku Java (2)

- Przeciążenie (*overloading*) metody. Taka sama semantyka jak w obiektowości (w UML).

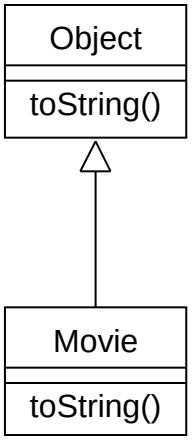
```
public float getPrice() {
    return price;
}
public float getPrice(float vatRate) {
    return price * (1.0f + vatRate / 100.0f);
}
```

- Przesłonięcie (*overriding*) metody. Taka sama semantyka jak w obiektowości (w UML).

```
public class Movie {
    // [...]
    @Override
    public String toString() {
        return „Movie: " +
        title;
    }}
```

Ekstensja klasy Film:
mt.mas.Film@126804e
4c3

Ekstensja klasy Film:
Movie: Terminator 1
Movie: Terminator 2



Trwałość ekstensji

- Ekstensja klasy jest trwała gdy jej obiekty „przeżyją” wyłączenie systemu – po ponownym włączeniu systemu będziemy mieli te same obiekty.
- W językach
  - Java,
  - MS C#,
  - C++

  cecha ta nie występuje bezpośrednio.
- W związku z tym implementujemy ja ręcznie zapamiętując ekstensje na dysku, a następnie je wczytując.

Trwałość ekstensji (2)

- W efekcie, zamiast tych samych obiektów, mamy takie same obiekty.
- Implementacja
  - Ręczna,
    - Szybkość,
    - Duża kontrola nad efektem końcowym,
    - Większa odporność na zmiany w kodzie utrwalanych klas,
    - Mały plik,
    - Wymaga (przeważnie) sporo pracy.

Trwałość ekstensji (3)

- Implementacja – *kont*.
  - Korzystająca z serializacji,
    - Bardzo łatwa w użyciu,
    - Mniejsza szybkość,
    - Duży plik,
    - Słaba odporność na zmiany w kodzie,
    - Częściowa możliwość kontroli dzięki:
      - dodaniu metod:
        - `private void writeObject(ObjectOutputStream stream) throws IOException`
        - `private void readObject(ObjectInputStream stream) throws IOException, ClassNotFoundException`
      - słowu kluczowemu `transient`.

Trwałość ekstensji (4)

- Implementacja – c. d.
  - W oparciu o bazę danych,
    - Konieczność mapowania struktur języka Java na konstrukcje z bazy danych,
    - Możliwość skorzystania z języka zapytań,
    - Praca z różnymi bazami danych,
    - Duży plik (czasami),
    - Szybkość działania
  - Korzystając z gotowych bibliotek (np. ORM).
    - Hibernate (http://www.hibernate.org/)
    - Java Persistence API (https://glassfish.dev.java.net/)
    - Java Data Objects (http://www.jpox.org/)
    - …

Trwałość ekstensji – implementacja ręczna

- W każdej z klas biznesowych znajduje się metoda zapisująca oraz odczytująca stan pojedynczego obiektu.

```
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);
   }

   // [...]
}
```

Trwałość ekstensji – implementacja ręczna (2)

- W każdej z klas zarządzających ekstensją (lub w klasie biznesowej jeżeli wybraliśmy takie podejście), znajdują się metody zapisujące oraz odczytujące całą ekstensję.

```
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);
      }
   }
}
```

Trwałość ekstensji – implementacja ręczna (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();
}
```

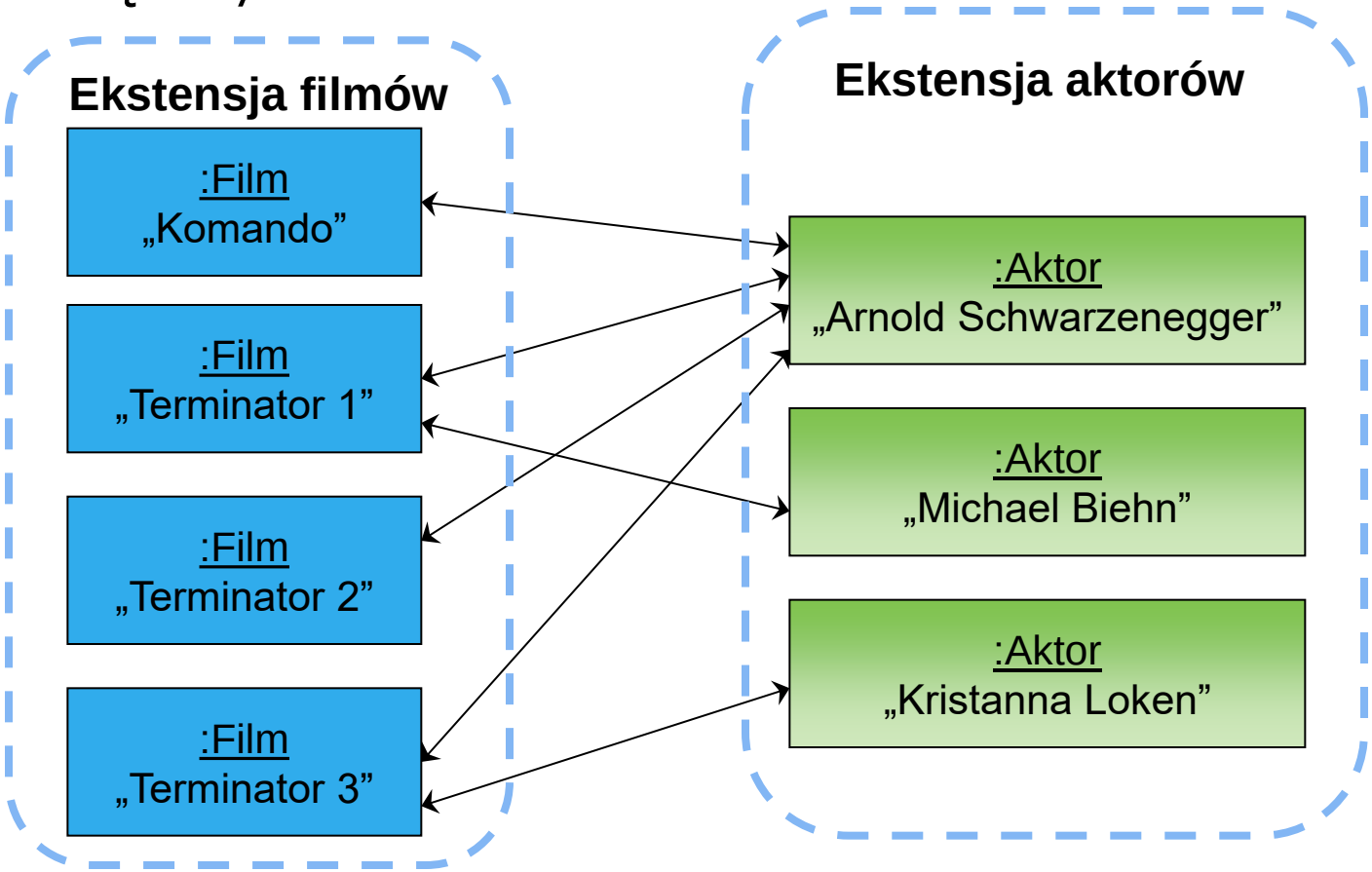
Trwałość ekstensji – implementacja ręczna (4)

- Przykład pokazuje zapis oraz odczyt ekstensji.
- Dzięki podzieleniu funkcjonalności na odpowiednie metody, możemy w łatwy sposób zapisywać inne ekstensje do tego samego strumienia (pliku).
- Dzięki temu, stan całej aplikacji może być zapamiętany w jednym pliku.
- Efekt działania
  - Ekstensja utworzona w pamięci,
  - Ekstensja zapisana i odczytana z pliku

```
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
```

Trwałość ekstensji – problem z powiązaniami

- Przedstawiony sposób ręcznej implementacji trwałości jest bardzo prosty. Nie pozwala na właściwe traktowanie powiązanych obiektów.



Trwałość ekstensji – implementacja z wykorzystaniem serializacji

- Serializacja jest mechanizmem zaimplementowanym w ramach bibliotek języka Java.
- Umożliwia automatyczne:
  - zapisywanie grafu obiektów do strumienia,
  - Odczytywanie grafu obiektów ze strumienia.
- Jedynym wymogiem, który trzeba spełnić aby z niego korzystać, jest „specjalna” implementacja przez klasę (oraz wszystkie jej elementy składowe) interfejsu **`Serializable`**.
- Specjalność implementacji interfejsu polega na tym, że w najprostszym przypadku deklarujemy jego implementację przez klasę, ale nie musimy implementować jego metod. Tym „zajmie się” kompilator języka Java.

Trwałość ekstensji – implementacja z wykorzystaniem serializacji (2)

- Deklarujemy implementację interfejsu przez klasę biznesową

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

   private String title;
   private float price;
   private LocalDate additionDate;  // requires Java 8+
}
```

- Tworzymy metody do zapisu oraz odczytu całej ekstensji

```
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();
   }
}
```

- Wszystkie ekstensje (wszystkich klas) **muszą być zapisane do tego samego strumienia** (pliku).

Trwałość ekstensji – implementacja z wykorzystaniem serializacji (3)

- Użycie

```
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(); }
```

- Rozmiar pliku z ekstensją:
  - Ręczna implementacja: 56 bajtów,
  - Serializacja: 263 bajty.
  - Dla większych danych różnice są mniejsze – około 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
```

Zarządzanie ekstensją

- Przedstawione sposoby implementacji zarządzania ekstensją będą (prawie) takie same dla każdej biznesowej klasy w systemie.
- Czy da się to jakoś zunifikować? Aby nie pisać wiele razy (prawie) tego samego kodu?
- Oczywiście – wykorzystamy dziedziczenie istniejące w języku Java.
- Opcjonalne rozwiązanie: klasy szablonowe (*Java Generics*) – **praca domowa** dla chętnych.

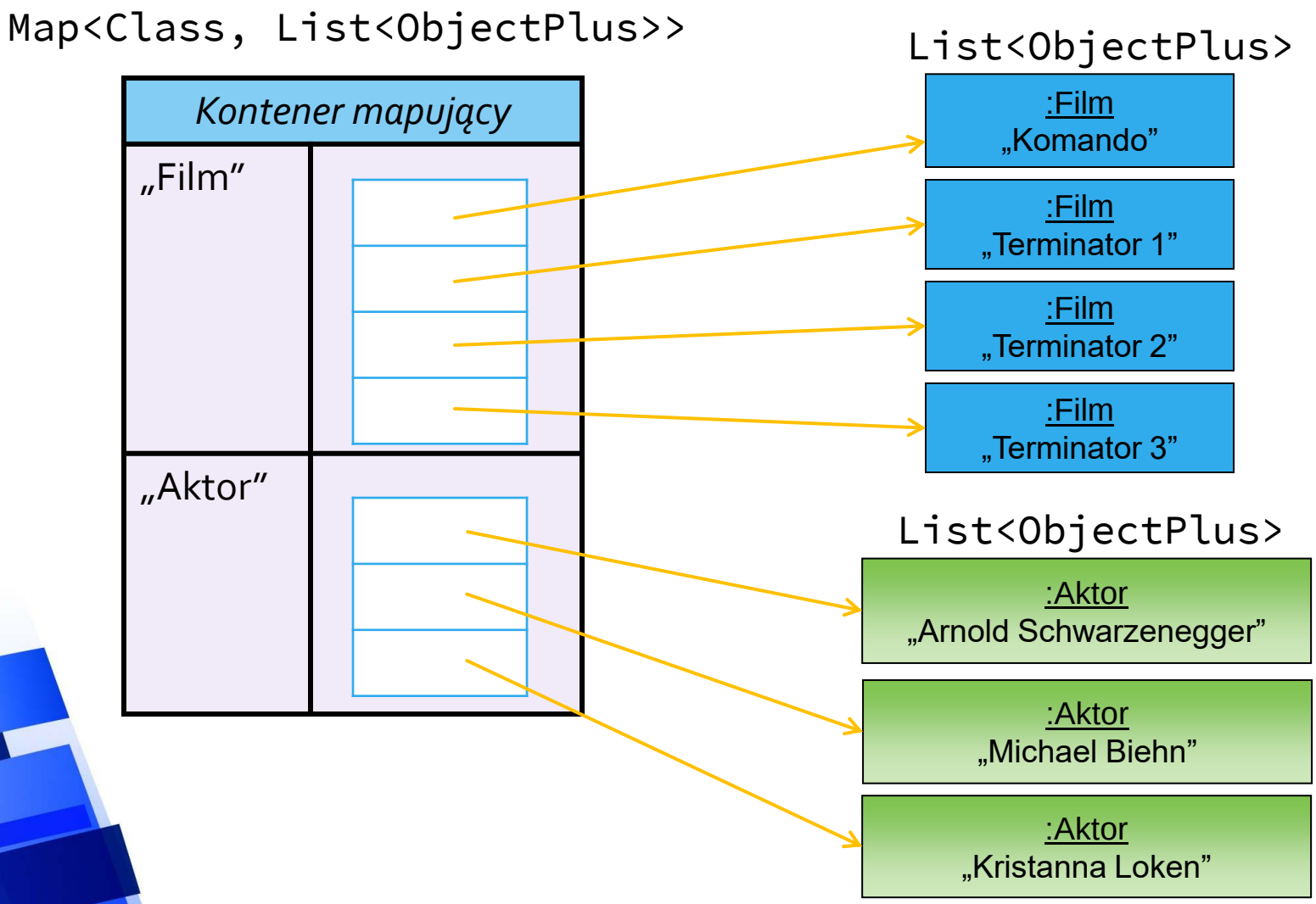
Uniwersalne zarządzanie ekstensją

- Stworzymy klasę z której będą dziedziczyć wszystkie biznesowe klasy w naszej aplikacji.
- Nazwijmy ją `ObjectPlus` i wyposażmy w:
  - trwałość,
  - zarządzanie ekstensją,
  - ?
- Zastosujemy pierwsze z omawianych podejść do implementacji ekstensji: w ramach tej samej klasy.

Uniwersalna ekstensja

- Ponieważ wszystkie biznesowe klasy dziedziczą z jednej nadklasy (*ObjectPlus*), nie możemy zastosować zwykłego kontenera przechowującego referencje.
- Użyjemy kontenera (mapy) przechowującego klucze i wartości:
  - Kluczem będzie nazwa konkretnej biznesowej klasy,
  - Wartością kontener zawierający referencje do jej wystąpień (właściwa ekstensja).
- Innymi słowy, ten nowy kontener będzie zawierał wiele ekstensji, a nie jedną ekstensję.

Uniwersalna ekstensja (2)



Uniwersalna ekstensja (3)

- Konstruktor każdej z klasy biznesowych będzie odwoływał się do konstruktora z nadklasy.

```
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
}
```

Uniwersalna ekstensja (4)

- Konstruktor z nadklasy zadba o właściwe dodanie do ekstensji.

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

   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);
   }

   // [...]
}
```

Uniwersalna ekstensja (5)

- Konstruktor z nadklasy zadba o właściwe dodanie do ekstensji – ***wersja skrócona*** `computeIfAbsent`.

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

   public ObjectPlus() {
      Class<? extends ObjectPlus> theClass = this.getClass();

      allExtents.computeIfAbsent(theClass, k -> new ArrayList<>()).add(this);
   }

   // [...]
}
```

Uniwersalna ekstensja (6)

- Utrwalenie takich ekstensji też jest bardzo proste (korzystamy z serializacji).

```
public class ObjectPlus implements Serializable {
   private static Map<Class<? extends ObjectPlus>, List<ObjectPlus>>
                     allExtents = new HashMap<>();
   // [...]

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

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

   // [...]
}
```

Pobranie ekstensji

- Wykorzystujemy metodę generyczną
- Zwracamy typ `Iterable<T>`

```
public class ObjectPlus implements Serializable {
    private static Map<Class<? extends ObjectPlus>, List<ObjectPlus>>
                        allExtents = new HashMap<>();
   // [...]

   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()));
   }}
```

- Wynik jest konkretnego typu, np. `Iterable<Movie2>`

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

Uniwersalne metody klasowe

- Część pomocniczych metod klasowych też może korzystać z ogólnej funkcjonalności zgromadzonej w nadklasie, np. wyświetlenie ekstensji.

```
public class ObjectPlus implements Serializable {
    private static Map<Class<? extends ObjectPlus>, List<ObjectPlus>>
                        allExtents = new HashMap<>();
   // [...]

   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);
      }
   }
}
```

Uniwersalne metody klasowe (2)

- Wyświetlenie ekstensji w klasie biznesowej (korzysta z funkcjonalności zdefiniowanej w nadklasie).

```
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?

- Co można ulepszyć w przedstawionym rozwiązaniu (ObjectPlus) w omówionym zakresie tematycznym?

- Praca domowa dla chętnych (opis, kod źródłowy) nagrodzona dodatkowymi pkt. na egzaminie.

Podsumowanie

- Część pojęć z obiektowości występuje w popularnych językach programowania.
- Niestety, niektóre z nich istnieją w niepełnym zakresie lub nie ma ich w ogóle.
- W większości przypadków, nieistniejące konstrukcje można:
  - zaimplementować samodzielnie na kilka, różnych sposobów,
  - Obsłużyć korzystając z gotowych bibliotek.
- Całą funkcjonalność związaną z zarządzaniem ekstensją klasy, warto zgromadzić w specjalnej nadklasie.

Pliki źródłowe

- Pobierz pliki źródłowe do wszystkich wykładów MAS

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