Design and Analysis of Information Systems (MAS)

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

Lecture 08

Implementation of remaining UML constructs in Object-Oriented Programming Languages

Outline

- Discussion of different types of UML constraints.
- Implementation of the constraints in object-oriented programming languages:
  - Related to attributes
  - Subset
  - Ordered
  - Bag
  - History
  - Xor
  - Custom
- Summary

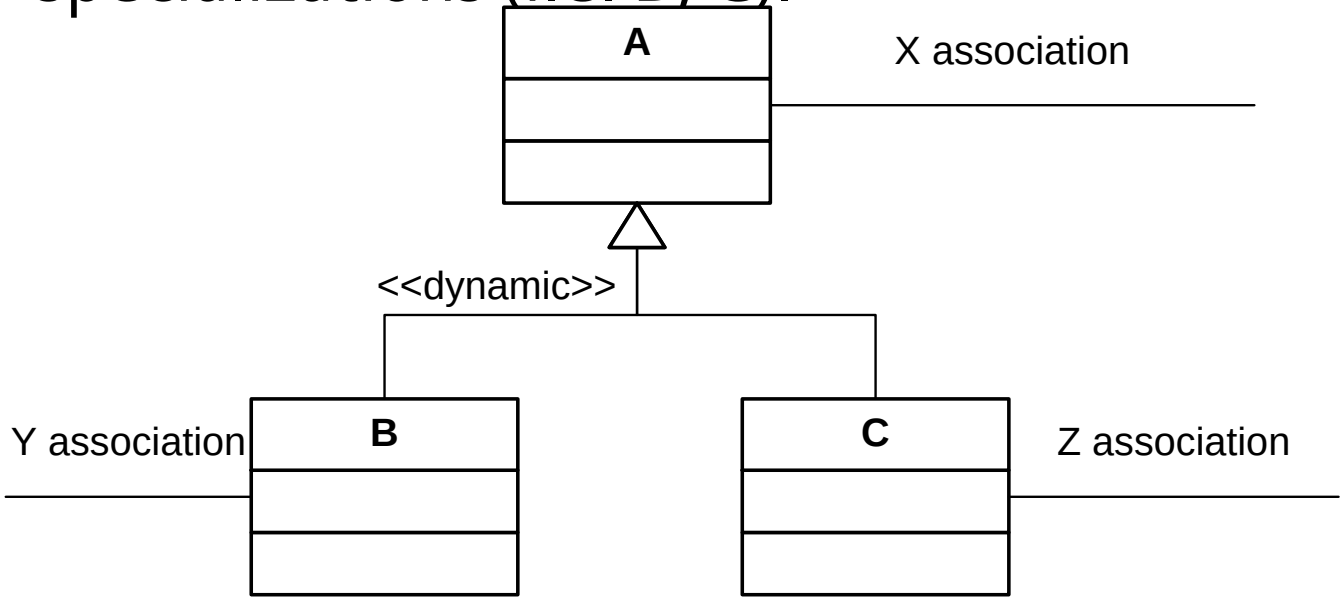
Homework

- Modification of the `ObjectPlusPlus` class improving implementation of the dynamic inheritance.

Students' presentations...

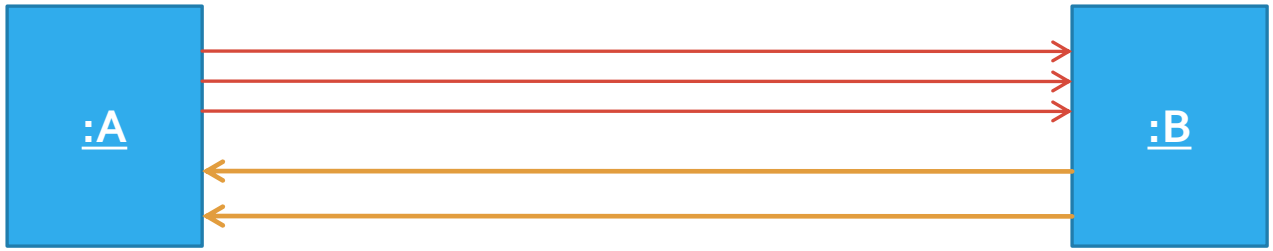
Homework (2)

- Additional issues
  - We should copy (from the „old" object) only links belonging to the super class (i.e. A). We do not copy links belonging to the specializations (i.e. B, C).



Homework (3)

- Additional issues – *cont*.
  - To make a proper update of the reference we need additional information about roles dependencies.
    - Which `roleName` is related to the `reverseRoleName`,
    - In theory we can replace all occurrences (in all roles) pointing to the „old" object, but some of them will not be updated (they should be abandoned) – see the previous slide.



Back to the current lecture

Constraints in the UML

- One of the extension mechanism of the UML.
- Makes possible refining of the model's semantics.
- Uses:
  - OCL
  - Mathematical/logical notations
  - Text
- There is some predefined constrain group (shipped with the UML).
- The notation: `{constrain content}`

Constraints Kinds

- **Dynamic**. During the validation process, the previous state of the item is taken into account.
- **Static**. During the validation process, the previous state of the item is not important.

Constraints Kinds (2)

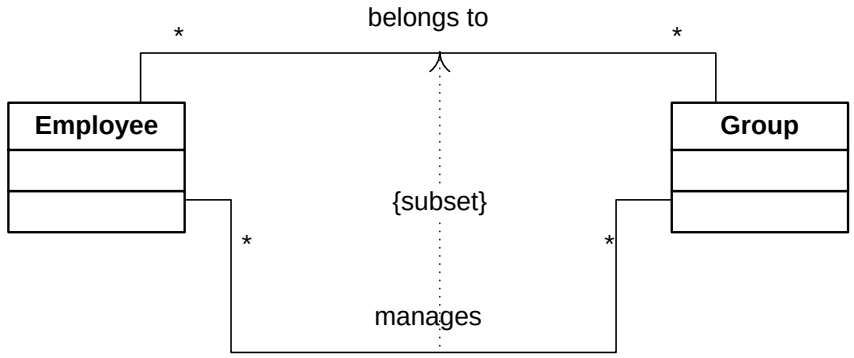
- Examples
  - A dynamic constraint
    - A salary cannot be decreased,
    - A salary increase has to be lower then 10%.
  - A static constraint
    - Salary > 2000
    - Salary < 5000

Predefined Constraints

- `{Unique}`
  - Imposed on an attribute in a class.
  - Assures its uniqueness in the entire extent.
  - Usually should allow to quickly retrieve the object based on the attribute.
  - Examples:
    - PESEL `{unique}`
    - NIP `{unique}`
    - SocialSecurityNumber `{unique}`
    - LastName `{unique}`

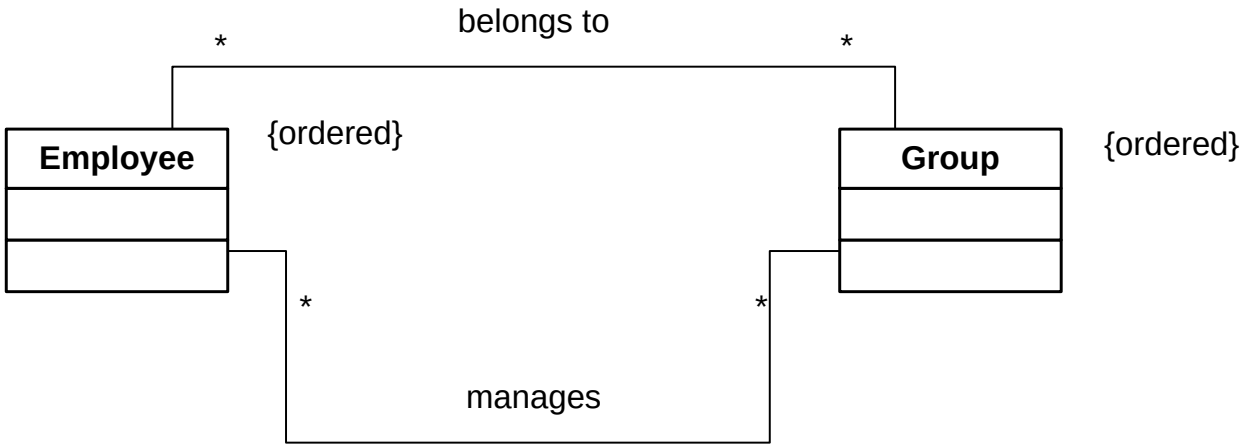
Predefined Constraints (2)

- {Subset}
  - Imposed on two associations (aggregations).
  - In order to create a link as an association B („manages"), there must be a link as an association A („belongs to").
  - Both associations should be among the same classes.
  - Moreover, both links have to be between the same objects.



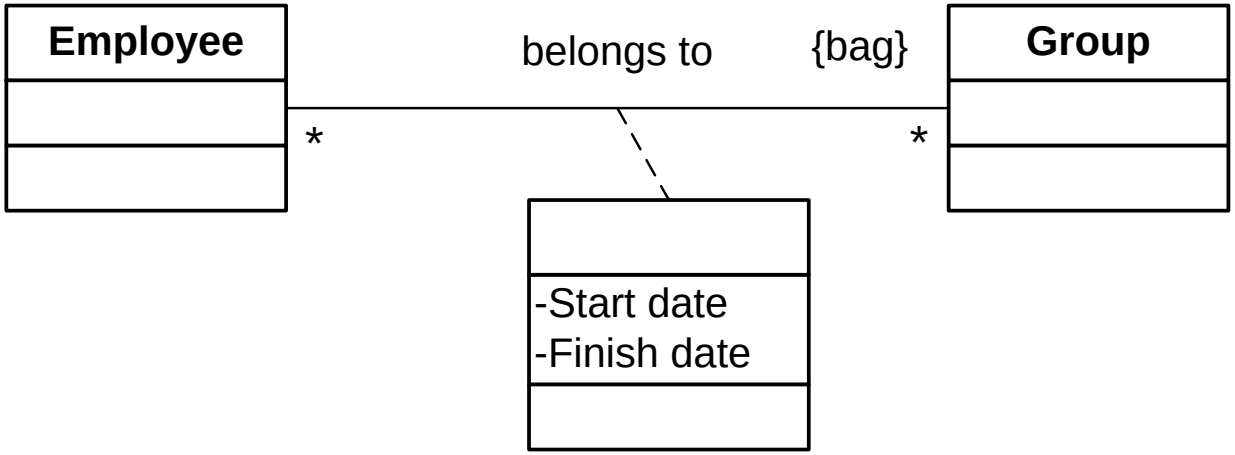
Predefined Constraints (3)

- `{Ordered}`
  - Imposed on:
    - **An association**. Means that links are stored (retrieved, processed) in some defined order.
    - **A class**. Objects in the extent are stored (retrieved, processed) in some defined order.



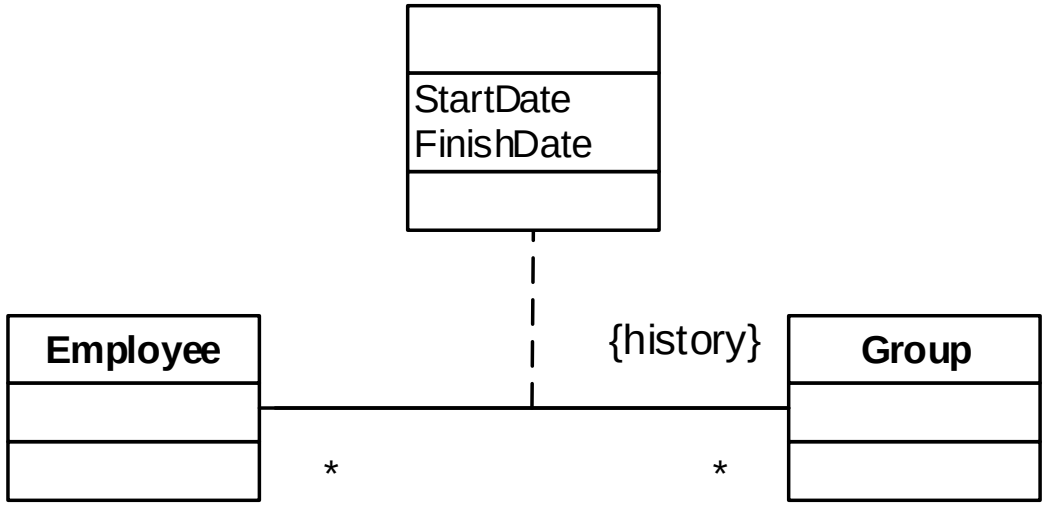
Predefined Constraints (4)

- {Bag}
  - Allows storing duplicated items.
  - In case of associations duplicated links can be stored (more than one link between the same objects).



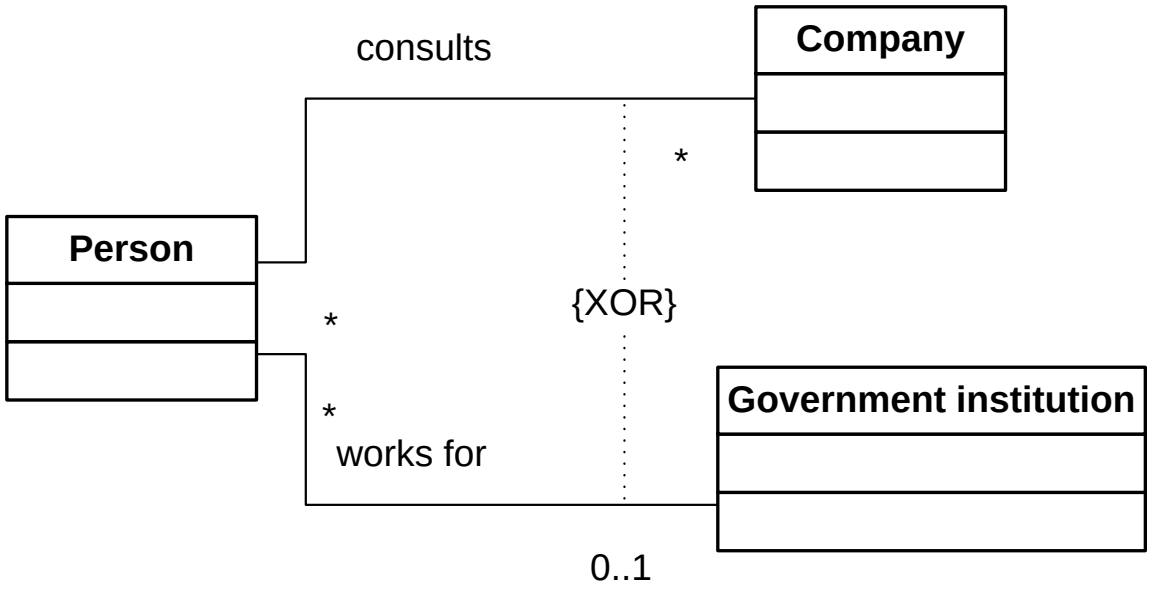
Predefined Constraints (5)

- `{History}`
  - Similar to the `{bag}`.
  - Duplicated links can be stored (more than one link between the same objects).
  - Emphasises time aspect of the connection.



Predefined Constraints (6)

- {XOR}
  - Imposed on at least two associations.
  - Guarantees that there will be only one link defined by the restricted associations.



The UML Constraints in Programming Languages

- In popular programming languages:
  - Java
  - C#
  - C++

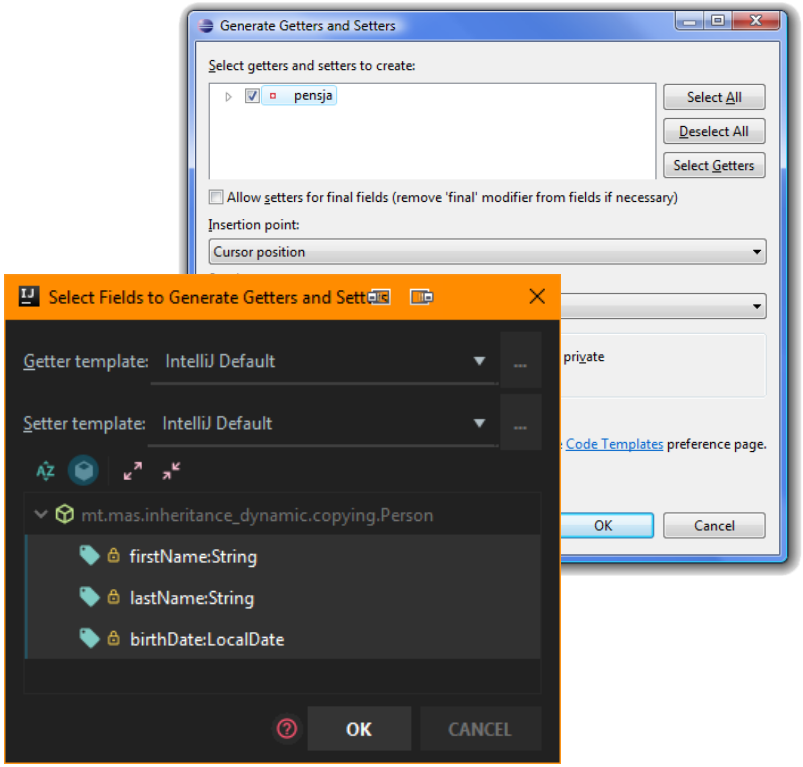
  constraints do not exist directly.
- However we can:
  - create dedicated methods which will follow them.
  - use:
    - a dedicated library – especially for the OCL,
    - other libraries, e.g. Hibernate, Entity Framework.
- The implementation complexity depends on the constraint kind.

Implementation of the attributes constraints

- Basic assumptions
  - All attributes in a class are hidden (e.g. as `private`),
  - All attributes activities are performed using dedicated methods:
    - get…
    - set…
  - The above rule should be also followed inside the attributes' class (unfortunately there is no way to force it).

Implementation of the attributes constraints (2)

- The modern programming environments (IDEs) support the process of creating necessary methods.
  - In the Eclipse there is a dedicated command: *Source* ➔ *Generate Getters and Setters*.
  - IntelliJ IDEA



Implementation of the attributes constraints (3)

- Watch out for:
  - giving the initial value (may be problematic due to percentage limitations of changes); take it into account in the dedicated `setXXX` method,
  - using defined values (constants) and utilizing them in all places of the program (also in error messages).

Implementation of the attributes constraints (3)

```
public class Employee {
   private float salary;

   public Employee(float salary) throws Exception {
      setSalary(salary);  // always use setters/getters
   }

   public float getSalary() {
      return salary;
   }

   public void setSalary(float salary) throws Exception {
      // Validate all the constraints
      if(salary < this.salary) {
         throw new Exception(String.format("The salary (%s) cannot be decreased!%d", salary));
      }

      if(this.salary != 0 && this.salary * (1d + maxSalaryChangePercentage /100d) < salary) {
         throw new Exception(String.format("The salary (%s => %s) increase cannot be more then %s%%",
                                    getSalary(), salary, maxSalaryChangePercentage));
      }

      if(salary < minimumSalary) {
         throw new Exception(String.format("The new salary (%s) has to be at least %s", salary, minimumSalary));
      }

      this.salary = salary;
   }

   @Override
   public String toString() {
      return String.format("Employee, salary: %s", getSalary());  // always use setters/getters
   }

   public final static float minimumSalary = 2000;
   public final static float maximumSalary = 5000;
   public final static int maxSalaryChangePercentage = 10;
}
```

Implementation of the `{unique}` constraint

- We need to take care for the uniqueness (in the class extent) of particular object's value, i.e. PESEL.
- It is possible to achieve this by providing a dedicated ***setter*** and for example:
  - Adding a class method which checks entire extent for existing values (a poor performance);
  - For every attribute marked with the `{unique}` we add a class attribute of type `Set` storing all values (from all objects of the extent);
  - Adding a single class attribute of type `Map` storing keys being names of the unique attributes and values of type `Set` storing all existing values;

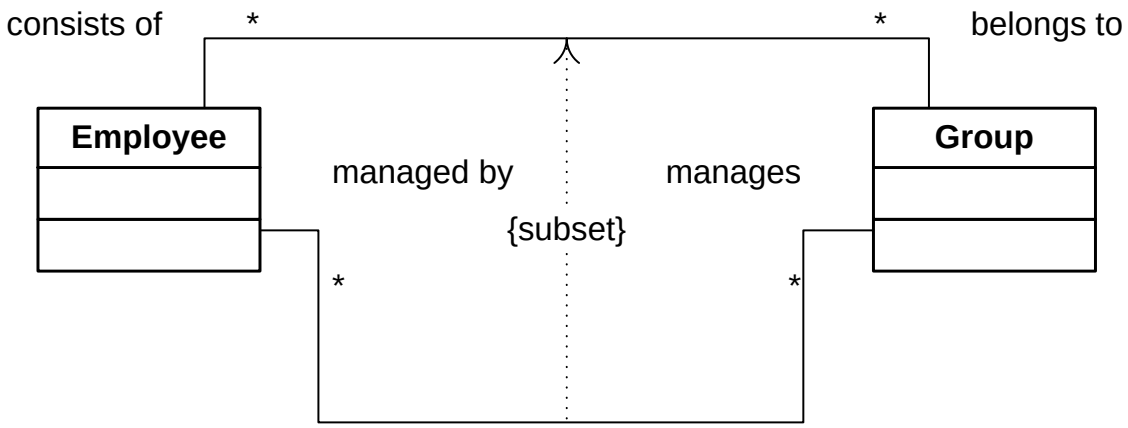
Implementation of the `{unique}` constraint (2)

- It is possible to achieve this by providing a dedicated ***setter*** and for example – *cont.*:
  - using a map, which:
    - aside of key **uniqueness**,
    - gives a way of **fast accessing** an object based on the attribute's value.

    This could be important because in many cases, marking an attribute as unique implies necessity for the above functionality.
- Pros and cons

Implementation of the `{subset}` constraint

- Links of roles „*consists of*", „*belongs to*" added regularly.
- Before creating a link described by the „*managed by*", „*manages*" roles, check if there is a link to the adding object in the „super" association.



Implementation of the `{subset}` constraint (2)

- We are going to modify the `ObjectPlusPlus` class
  - We add a method which check if there is a link to a given object in the given role.
- Let's create a new class: `ObjectPlus4`
  - Inheriting from the `ObjectPlusPlus`
  - Containing the functionality related to constrains .

Implementation of the {subset} constraint (3)

- A method in the ObjectPlusPlus, which checks if there is a link to a given object as a given role.

```
public abstract class ObjectPlusPlus extends ObjectPlus implements
Serializable {
   private Map<String, Map<Object, ObjectPlusPlus>> links = new
Hashtable<>();

   // [...]

   public boolean isLink(String roleName, ObjectPlusPlus targetObject) {
      Map<Object, ObjectPlusPlus> objectLink;

      if(!links.containsKey(roleName)) {
         // No links for the role
         return false;
      }

      objectLink = links.get(roleName);

      return objectLink.containsValue(targetObject);
   }
}
```

Implementation of the {subset} constraint (4)

- A method in the ObjectPlus4, which adds a new link using the {subset} constraint.

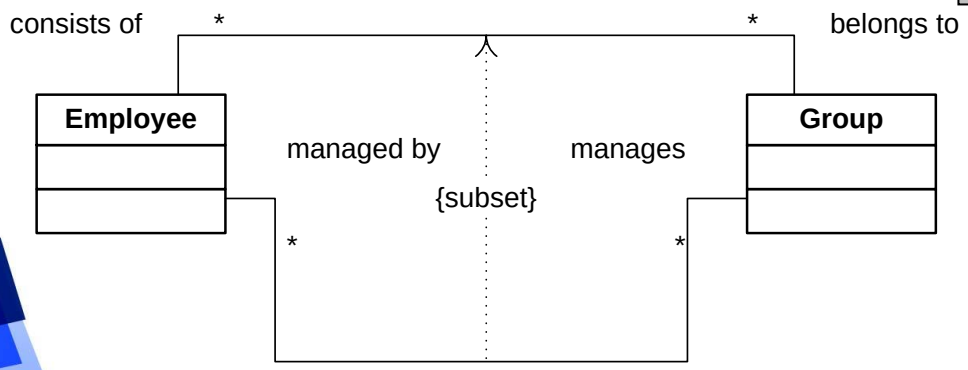
```
public abstract class ObjectPlus4 extends ObjectPlusPlus {

   public void addLink_subset(String roleName, String reverseRoleName,
          String superRoleName, ObjectPlusPlus targetObject) throws Exception {

      if(isLink(superRoleName, targetObject)) {
         // There is a (super) link to the added object in the super role
         // Create the link
         addLink(roleName, reverseRoleName, targetObject);
      }
      else {
         // No super link ==> exception
         throw new Exception("No link to the '" + targetObject + "' object in the ,"
                    + superRoleName + "' super role!");
      }
   }

   // [...]
}
```

Implementation of the {subset} constraint (5)

- Sample case – an automatic version

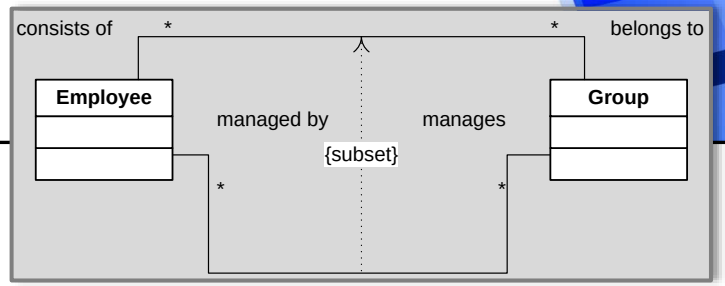
```
private static void testSubset_auto() {
   var employee = new Employee("John", "Smith");
   var group = new Group("Group no 1");

   try {
      // Add the ordinary link
      employee.addLink(Employee.roleBelongsTo, Group.roleConsistsOf, group);

      employee.addLink_subset(Employee.manages, Group.roleManagedBy, Employee.roleBelongsTo,
group);

      // Show links info
      employee.showLinks(Employee.roleBelongsTo, System.out);
      employee.showLinks(Employee.manages, System.out);

   } catch (Exception e) {
      // [...] }}
```

```
Employee links, role 'belongs to':
  Group: Group no 1
Employee links, role 'manages':
  Group: Group no 1
```



Implementation of the {subset} constraint (5)

- Sample case – a manual version




```
private static void testSubset_manual() {
   var employee = new Employee("John", "Smith");
   var group = new Group("Group no 1");

   try {
      // Add ordinary link
      employee.addLink(Employee.roleBelongsTo, Group.roleConsistsOf, group);

      // Check if there is a super link
      if(employee.isLink(Employee.roleBelongsTo, group)) {
         // There is a super link => add a subset link
         employee.addLink(Employee.manages, Group.roleManagedBy, group);
      }
      else {
         // No super link
         System.out.println("No super link for the role: " + Employee.roleBelongsTo);
      }

      // Show links info
      employee.showLinks(Employee.roleBelongsTo, System.out);
      employee.showLinks(Employee.manages, System.out);

   } catch (Exception e) {
      // Error
      e.printStackTrace();
   }
}
```

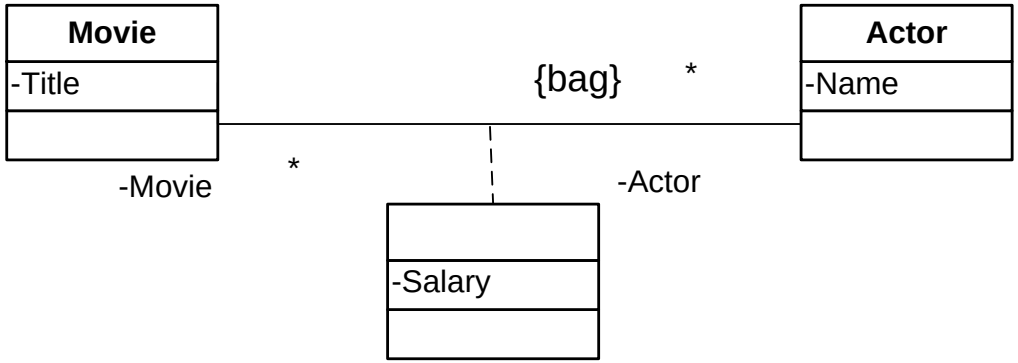
```
Employee links, role 'belongs to':
   Group: Group no 1
Employee links, role 'manages':
   Group: Group no 1
```

Implementation of the `{ordered}` constraint

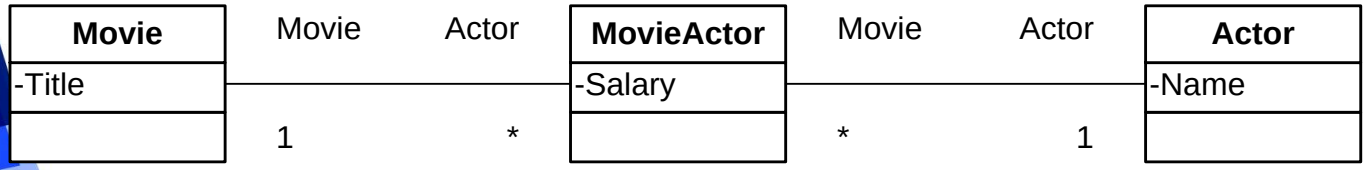
- For an association
  - We need to use a container which guarantees the items' order. In case of:
    - Adding order – i.e. `List`
    - Custom order, i.e. `TreeSet` together with a dedicated comparator (`Comparator`).
- For an extent
  - Similarly to the association.
- The `ObjectPlusPlus` class uses the `ArrayList` class for an extent (ordered) and the `HashMap` for links.

Implementation of the {bag} constraint

- In case of associations we need a container allowing storing duplicated objects (identified i.e. by a hash or a reference).
- Duplicates are usually stored when we use an association attribute providing additional information about a link.



- However in such a case we introduce a middle class anyway. Hence there will be no duplicated references.



Implementation of the {bag} constraint (2)

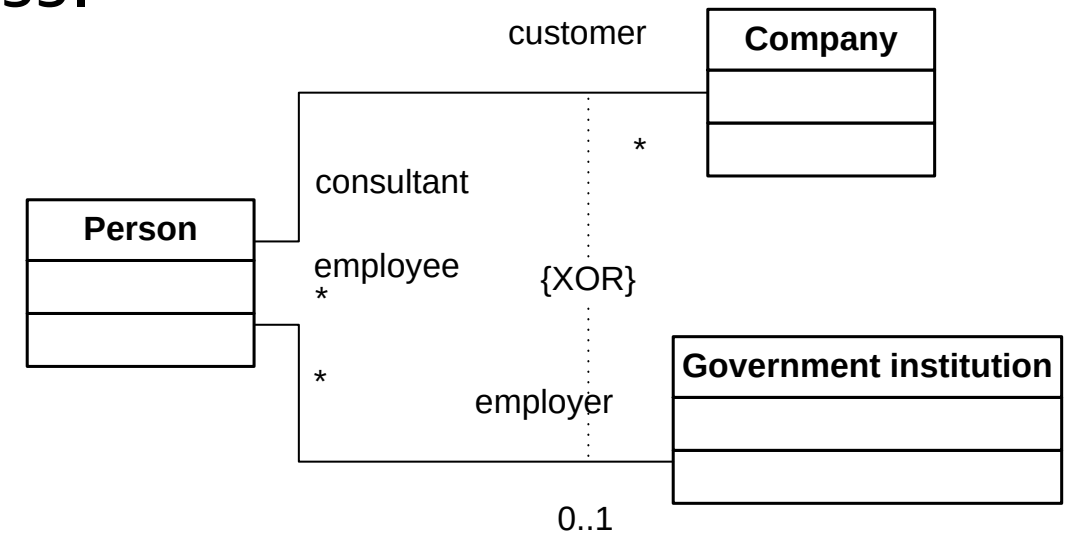
- But if somebody would like to store real duplicates than a proper container has to be chosen, e.g. `Vector`.
- The `ObjectPlusPlus` class uses the `HashMap` class which takes care about unique keys. Hence it is possible to add a duplicated value but with a unique key.

Implementation of the {history} constraint

- Does the {history} constrain mean any special consequences comparing to the {bag} constraint?
- It seems that it does not.
- Conclusion: the implementation of the {history} constraint is similar to the {bag}.

Implementation of the {XOR} constraint

- We need to make sure that in the defined roles (associations) group there will be only one link.
- We use a modified `ObjectPlus4` class.



Implementation of the {XOR} constraint (2)

```
public abstract class ObjectPlus4 extends ObjectPlusPlus {

   private List<String> rolesXOR = new LinkedList<>();

   public void addXorRole(String xorRoleName) {
      rolesXOR.add(xorRoleName);
   }

   public void addLinkXor(String roleName, String reverseRoleName, ObjectPlusPlus
                     targetObject) throws Exception {
      if(rolesXOR.contains(roleName)) {
         // The currently adding role is XOR'ed

         // Check if there is a link for XOR'ed roles
         if(isXorLink()) {
            throw new Exception("There is a link for a XOR roles!");
         }

         // There is no link ==> add a link using an already existing method from
         // a super class
      }

      // Add the link
      super.addLink(roleName, reverseRoleName, targetObject);
   }
   // [...]
}
```

Implementation of the {XOR} constraint (3)

```
public abstract class ObjectPlus4 extends ObjectPlusPlus {

   private List<String> rolesXOR = new LinkedList<>();

   // [...]

   private boolean isXorLink() {
      for(String role : rolesXOR) {
         if(this.anyLink(role)) {
            return true;
         }
      }

      return false;
   }
}
```

- Optionally to do:
  - Checking for the {xor} constraint during adding a link using the „reverse side” (class).

Implementation of custom constraints

- In case of custom, other constraints, e.g. written with a „plain text”
  - Unfortunately we need to understand what was the author’s intention,
  - Implement it using proper methods, approaches, etc (no out-of-the-box solutions).
  - Sometimes there might be a need of changing the entire approach for creating the system, e.g. by using an orthodox encapsulation.

The Summary

- In popular programming languages the constraints do not exist directly.
- Some kinds of them could be implemented using an existing solutions, i.e.
  - For attributes,
  - Predefined: `{subset}`, `{xor}`.
- In a general case (constrains written with a plain text) there is a necessity for a manual implementation.

Source files

Download source files for all MAS lectures

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