Deel 4: Associaties

Class diagrams kunnen uitleggen.

Class diagrams provide an overview of classes and their associations, attributes[1] and methods. Class diagram encapsulate the implementation.

Side issue
Object diagrams are another type of diagram supported by UML[2]. They involve objects, linked with other objects, values of attributes and so on.
There are two principles defined for classification. These rules are temporary as inheritance will solve this problem.

1. **Static** classification
   Each object is an instance of the same class for its entire lifetime.
2. **Disjoint** classification (=dynamic classification)
   Classes should be specific enough in order to allo¡ for each object to fit exactly in one class. See coding rule 77 for manipulation.

Het concept uni-directional association kunnen uitleggen.

If there is a uni-directional association from one class to another, the former one will provide constructors for initializing the associated properties, and provide inspectors and mutators for them. (example BankAccount and Person)
Uni-directional associations are stored as a variable with the referred clas as its type.
Remark: When invariants may become violated upon a change of the associated object, it'll better to design the underlying relation as a bidirectional association.

Het concept bi-directional association kunnen uitleggen.

If we require functionalities in both directions, or if this is expected to happen in the future, we'll need bi-directional associations. Navigation is supported in both directions, as bi-directionally associated objects are aware of one another.

Remark: Give the control over to the least complex of both participating classes. A simple measure for the complexity is the amount of methods of a class.

[1] public variable inside the class/object
[2] graphical language that offers several types of diagrams to document different aspects of software systems

Lists, sets and maps kunnen uitleggen.

When implementing associations with unrestricted multiplicity, we use the collections framework to store references to objects with which the prime object is associated. Collections are either lists, sets and maps. In a *list*, the order is important. In a *set*, the elements are not positionally ordered and it cannot have multiple occurrences of the same element. *Maps* are data structures in which keys are mapped onto values.

Lists	Sets	Maps
▪ **ArrayList** The array lists grow and shrink according to the number of elements they contain ▪ **LinkedList** Elements are explicitly linked to each other. ▪ **Vector** Offers same methods as ArrayList. (old version)	▪ **HashSet** Do not always return their elements in the same order. Allows to easily and quickly determine whether an object is already in the set or not. ▪ **LinkedHashSet** Returns elements always in the same order. ▪ **TreeSet** Stores elements in some ascending order.	▪ **HashMap** Same as ArrayList. ▪ **LinkedHashMap** Same as LinkedList. ▪ **TreeMap** Same as Vector.

Remark: A set cannot have multiple occurrences of the same element.