KATHOLIEKE UNIVERSITEIT LEUVEN

Department of Computer Science

Associations with Restricted Multiplicity

- Bi-directional Associations
- Destructors
- Memory management

Assignment

KATHOLIEKE UNIVERSITEIT LEUVEN

- Develop a definition of a class of persons involving (only) the following characteristics:
  - The gender of a person
  - The spouse of a person
- Prior to specification and implementation, a design must be worked out identifying:
  - Classes involved in the system
  - Associations between objects of the identified classes
  - Attributes and methods applicable to objects of the identified classes

Overview

KATHOLIEKE UNIVERSITEIT LEUVEN

- Object-Oriented Design
  - Identify classes, associations, attributes and methods
    - Draw class diagrams in the Unified Modeling Language (UML)
- Enumerations
  - An enumeration defines a set of named objects
- Associations
  - Uni-directional or bi-directional associations
    - For bi-directional associations, consistency is an important issue
- Destructors
  - Classes should manage the entire lifetime of its objects
    - Introduce destructors for classes with real-world semantics
- Epilogue

Class Diagrams

KATHOLIEKE UNIVERSITEIT LEUVEN

- Classes are represented as rectangles involving 3 compartments
  - The top compartment shows the name of the class
  - The middle compartment enumerates attributes (properties) ascribed to the class and its objects
  - The bottom compartment lists methods applicable to the class and to its objects
- Associations are represented by lines connecting the classes involved
  - Associations may be loaded with an association name and with role names for each of the classes involved
    - Associations in UML are nearly always binary
  - Associations further reveal constraints of multiplicity at both ends
    - A multiplicity value of the form m..n states that a single object at the other end must be associated with at least m and at most n objects
  - Associations can be uni-directional or bi-directional
    - UML offers the notion of navigability at each end of the association

Task 1

Overview

KATHOLIEKE UNIVERSITEIT LEUVEN

- Object-Oriented Design
  - Identify classes, associations, attributes and methods
    - Draw class diagrams in the Unified Modeling Language (UML)
- Enumerations
  - An enumeration defines a set of named objects
- Associations
  - Uni-directional or bi-directional associations
    - For bi-directional associations, consistency is an important issue
- Destructors
  - Classes should manage the entire lifetime of its objects
    - Introduce destructors for classes with real-world semantics
- Epilogue

Enumerations

KATHOLIEKE UNIVERSITEIT LEUVEN

- Java 1.5 supports the definition of type-safe enumerations
  - An enumeration defines a fixed set of named objects
    - Instance variables and instance methods defined at the level of the type itself apply to all its elements
    - Instance variables and instance methods defined at the level of an individual element only apply to that element
  - Elements of an enumeration type are accessed in a qualified way
    - "EnumerationType.ELEMENT$_i$" denotes the named object "ELEMENT$_i$" of the given enumeration type

```
public enum EnumerationType {

   ELEMENT1 { /* Methods and variables for ELEMENT1 */ },
   …
   ELEMENTn { /* Methods and variables for ELEMENTn */ };

   /* Methods and variables common to all elements */
 }
```

Task 2

Overview

KATHOLIEKE UNIVERSITEIT LEUVEN

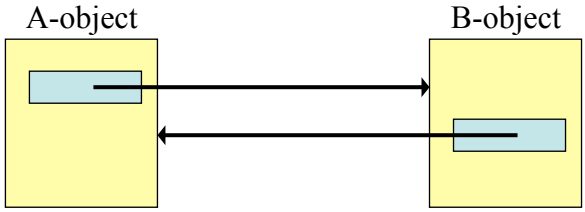
- Object-Oriented Design
  - Identify classes, associations, attributes and methods
    - Draw class diagrams in the Unified Modeling Language (UML)
- Enumerations
  - An enumeration defines a set of named objects
- Associations
  - Uni-directional or bi-directional associations
    - For bi-directional associations, consistency is an important issue
- Destructors
  - Classes should manage the entire lifetime of its objects
    - Introduce destructors for classes with real-world semantics
- Epilogue

Associations

KATHOLIEKE UNIVERSITEIT LEUVEN

Task 3

- Bi-directional associations require class invariants imposing consistency
  - General restrictions on associated objects are typically worked out in the inspector "canHaveAsα(T)"
    - This inspector checks whether its prime object could ever be associated with the given object.
  - Consistency of bindings for bi-directional associations is worked out in the inspector "hasProperα()"
    - This inspector encapsulates the class invariant, and appeals to the inspector canHaveAsα



Methods for Associations

KATHOLIEKE UNIVERSITEIT LEUVEN

- Introduce getters returning information concerning the involvement of an object in a bi-directional association
  - Getters must be introduced in both classes
- Introduce setters for registering associations among objects
  - Both setters must be accessible in at least one of the classes involved
    - No setters are introduced for immutable ends
- Introduce constructors to set up associations at the time new objects are created
  - Public constructors may have to change the state of other objects, in order to guarantee consistency
- Introduce public mutators to set up and break down associations
  - At least one class will introduce public mutators, except for immutable associations

Task 4+5+6+7

Overview

KATHOLIEKE UNIVERSITEIT LEUVEN

- Object-Oriented Design
  - Identify classes, associations, attributes and methods
    - Draw class diagrams in the Unified Modeling Language (UML)
- Enumerations
  - An enumeration defines a set of named objects
- Associations
  - Uni-directional or bi-directional associations
    - For bi-directional associations, consistency is an important issue
- Destructors
  - Classes should manage the entire lifetime of their objects
    - Introduce destructors for classes with real-world semantics
- Epilogue

Garbage Collection

KATHOLIEKE UNIVERSITEIT LEUVEN

- In Java, the object memory is managed by the garbage collector
  - The garbage collector reclaims the memory occupied by all objects that are no longer in reach
    - An object is directly reachable, if some local variable or some formal argument references it
    - An object is indirectly reachable, if some instance variable of a reachable object or a static variable references the former object
- Garbage collection executes in a separate thread
  - The garbage collector can be invoked explicitly using the static method “gc” offered by the predefined class “System”

Object Life Time

KATHOLIEKE UNIVERSITEIT LEUVEN

- Java offers a predefined method “finalize” to deal with the destruction of objects
  - The method is applied to each object just before it is about to be cleaned up by the garbage collector
    - Each class may introduce its own version of “finalize”, working out some final actions before the object actually disappears
    - Java suggests to use “finalize” to release resources such as files and network connections, that are still occupied by such objects
- Important objects are best destroyed explicitly, reflecting events applied to them in the real-world
  - Bank accounts, room reservations, … must not be destroyed implicitly by taking away the last reference to them

Destructors

KATHOLIEKE UNIVERSITEIT LEUVEN

- A class may introduce a single destructor, destroying (terminating) the object to which it is applied
  - Each destructor is named “terminate”, has no arguments and does not return a result
    - A destructor may throw exceptions
- A class may introduce a single inspector, returning an indication whether or not an object has been destroyed
  - The inspector is a basic inspector named “isTerminated”, has no arguments and returns a boolean
- Terminated objects must still satisfy all their invariants
  - Invariants concerning properties ascribed to individual objects may distinguish between terminated and non-terminated objects
  - Mutators must deal with the exceptional case in which at least one of the involved objects is terminated

Task 8

Overview

KATHOLIEKE UNIVERSITEIT LEUVEN

- Object-Oriented Design
  - Identify classes, associations, attributes and methods
    - Draw class diagrams in the Unified Modeling Language (UML)
- Enumerations
  - An enumeration defines a set of named objects
- Associations
  - Uni-directional or bi-directional associations
    - For bi-directional associations, consistency is an important issue
- Destructors
  - Classes should manage the entire lifetime of its objects
    - Introduce destructors for classes with real-world semantics
- Epilogue

Conclusion

KATHOLIEKE UNIVERSITEIT LEUVEN

- Software development starts with a preliminary design
  - A class diagram identifies classes and associations
    - Each class is complemented with attributes and methods
- Consistency is an important issue in bi-directional associations
  - Mutators and constructors must manipulate references in both directions
    - The setter for a bi-directional association with restricted multiplicity just registers one side of the association
  - Conditions imposed on bi-directional associations are encapsulated in inspectors “canHaveAsα(T)” and “hasProperα()”
- Real-world classes are complemented with a destructor
  - Both a method “terminate” and an inspector “isTerminated” are best defined
    - All other methods must explicitly deal with terminated objects

Homework

KATHOLIEKE UNIVERSITEIT LEUVEN

- Extend the definition of the class of persons with a method to switch partners
  - The method must be such that it only throws an exception if the person to switch partners with is not effective