Package Diagrams

Massimo Felici

Room 1402, JCMB, KB

0131 650 5899

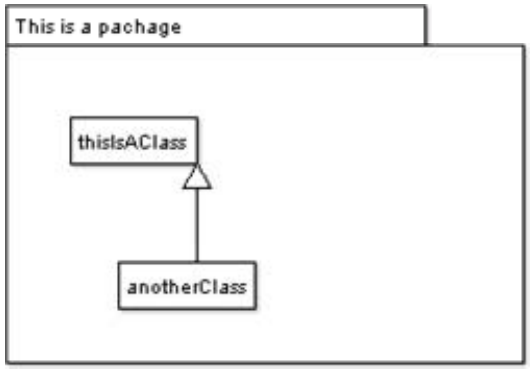
mfelici@inf.ed.ac.uk

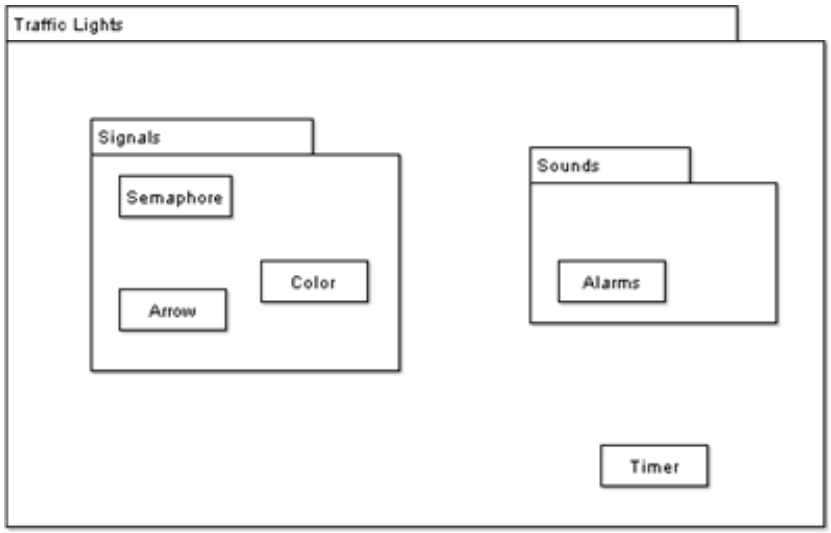
Rationale

- Provide a way to **group** related UML elements and to **scope** their names
- Provide a way to visualize **dependencies** between parts of your system
  - Vulnerable to **changes** (in other packages)
- Provide support for analysis
- Determine compilation order

- Package design needs to balance diverse needs
  - Easier to build and test
  - Better tracking and property transparency
  - Working at a stable overview without the noise of low-level details
  - Less conflict between distributed teams
  - Easy refactoring and extension

Representation of Packages





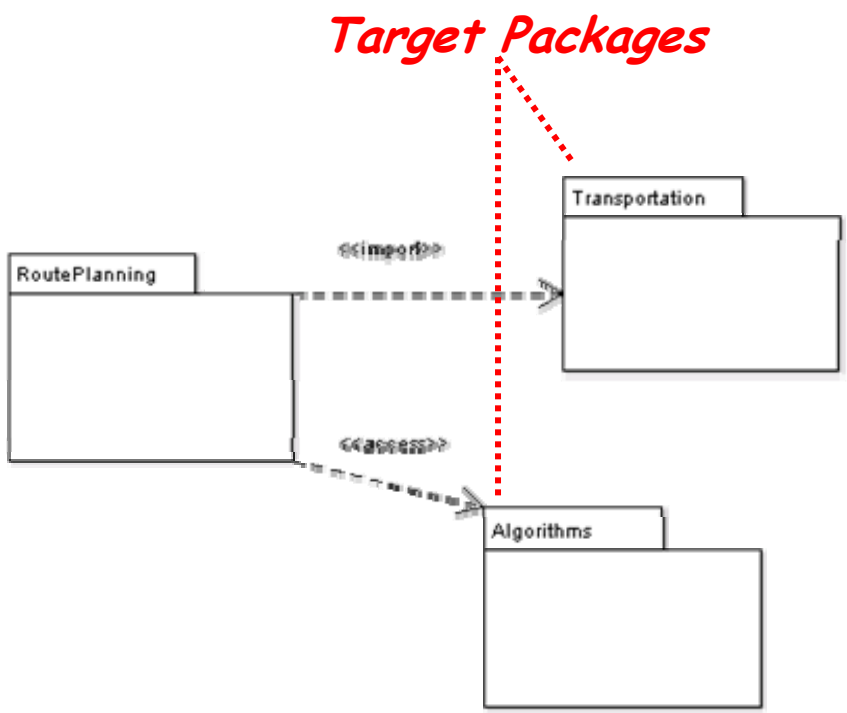
- Packages contain different elements (packages too)
- A UML package establishes a namespace
  - To specify the context of a UML, you provide the fully-scoped name
  - ***packageName::className***
  - In Java, a fully-scoped name corresponds to specify the Java package
- It is possible to specify **visibility** for owned and imported elements
  - **public** or **private**
  - No elements – no assumptions about the package's content

Element Visibility

- Elements with **public visibility** are accessible outside the package
- Elements with **private visibility** are available only to other elements inside the package
- In Java, public and private visibilities correspond to a class being public or private to a Java package
- If the public keyword is absent, then the class is private to the package

Importing and Accessing Packages

- **<<import>>**: Elements of imported packages are available without qualification in the importing package
  - public visibility
  - private visibility
  - Import specific elements, rather than the whole package
- **<<access>>**:Accessing packages whereas gives private visibility to the imported elements

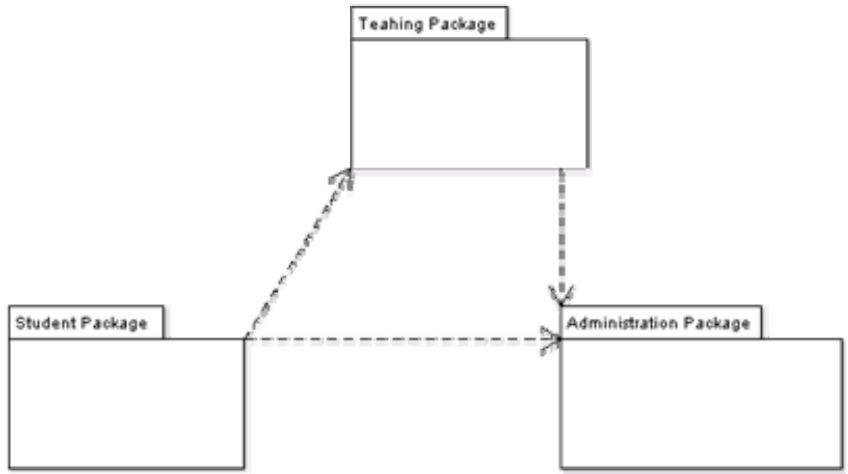


Merging Packages

- Creates relationships between classes of the same name
- Merge is a directed relationship
- **Rationale**: the evolution from UML 1.x to UML 2.0 - extending a base concept of elements without renaming
- Some Rules for package merge
  - Private members are not merged
  - Merging classes are generalized to corresponding merged ones
  - Maintain package scope for reference to classes
  - Classes outside the intersection of packages are unchanged
  - Subpackages are added, if they don't exist
  - Merge extends to subpachages with the same names
  - Merge acquires imported elements

Package Dependencies

- Structuring a Project with Packages
- Packages group UML elements and organize a logical system during design and implementation
- Manage Dependencies
  - Directed dependency graphs
  - Avoid **cyclical package dependencies**
  - Organize and allocate project work to different teams - Different groups can work on different packages without destabilizing each other



Use Case Packages

- Using packages to organize use cases
- Organize the functional behavior of a system
- Highlight which actors interact with which portions of the system