

CSE333 Examination Handout

Reusability Assumptions and Problem Background

The reusability assessment and refactoring process for classes that has been discussed is highly dependent on the generality assignment (marking) and the related classes (to be reused together in a future application) that have been defined by the software engineer. However, as applications get more and more complex, with more and more classes, the ability to easily mark generalities for each class becomes problematic. To prepare you for one of the final examination problems, consider Figure 1, where a set of classes is shown. In Figure 1, the inheritance hierarchy is in the middle, with empty boxes and boxes with X's, connected by inheritance with arrows that have solid filled in heads. The other classes that are not in the hierarchy are indicated by double boxes or diagonal shaded boxes. Related classes are indicated by open-headed arrows that are either dashed (from a double/shaded class to a class in the hierarchy) or solid (from a class in the hierarchy to a double/shade class). To set the context for this problem, assume the following:

- There is only single inheritance.
- There are m levels in the hierarchy.
- There are n levels of generality, with $n < m$.
- In each of the m levels of the inheritance hierarchy, at least one of the classes (siblings) has a preset generality level that CANNOT be changed.
- The root class of the inheritance hierarchy has a level of G_0 .
- At least one leaf class of the inheritance has a level of G_{n-1} .
- All of the related classes (with double boxes or diagonal shaded boxes) have preset generality levels that CANNOT be changed.
- None of the present generality levels cause any Type 3 dependencies and satisfy constraints within the inheritance hierarchy and between related classes and hierarchy classes.
- The only non-inheritance dependencies that exist in Figure 1 mirror the related classes. That is, if two classes are not related you can assume that there are NOT any dependencies (e.g., method calls, attribute inclusions, etc.) that would cause a bad dependency.
- Each related arrow has a dependency/coupling of 1, which means that even if there are multiple method calls, attribute, inclusions, etc., only 1 is to be counted.
- Each class is related to at most ONE other class, i.e., a class cannot be related to two or more classes.
- There are ONLY inheritance dependencies within the hierarchy - there are no dependencies among siblings or between a class and its non-parent ancestors and/or descendants.

To help you in the exam problem, it is useful to assume the following structures:

- Each class in Figure 1 can be represented by a C-like structure which contains `class_ID`, `generality_level` (from G_0 to G_{n-1} , set according to the conditions of assumptions), `parent_PTR` (null for root class), `sibling_PTR` (set to the sibling to the right of the class - last one null), `related_to_PTR` (from class to another class - may be null), and the `related_from_PTR` (to class from another class - may be null).
- The set of leaf nodes (classes), SLN, that are the boxes with arrows.

You are advised to study and familiarize yourself with these assumptions and background material prior to the final examination.