Overview

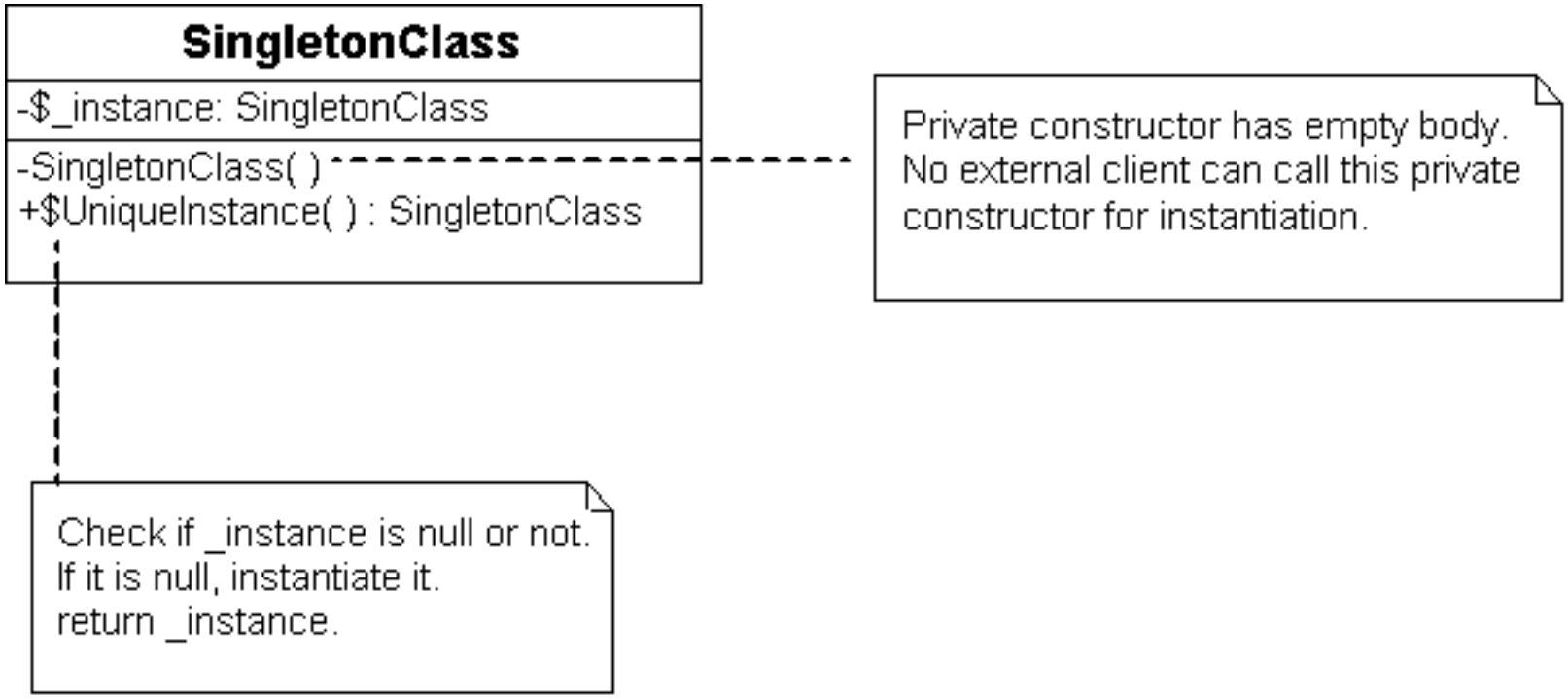
- The Singleton Pattern
- Nested and Inner Classes
- Some Philosophy

The Singleton Pattern

- Recall the EmptyListNode class that represents the "empty list". Conceptually, there is only one empty list in the "world".
  - The concept is akin to that of the empty set: there is only one empty set.
- How can we ensure that only one instance of EmptyListNode can be created throughout the life of a program?
- There is away to design a class to ensure such uniqueness property. It is called the *Singleton Design Pattern*.

The Singleton Pattern (cont.)

- The following UML diagram describes the pattern:



- Note: The field `_instance` and the method `UniqueInstance()` are of class scope (i.e. static).

The Singleton Pattern (cont.)

- The method `UniqueInstance()` is called a "factory" method as it is used to manufacture an instance, though unique, of the SingletonClass.

- The class SingletonClass is appropriately called a "factory". In this very special case, SingletonClass manufactures its own (unique) instance. (Recall that we saw the *Factory Pattern* in Lab 2.)

Nested and Inner Classes

- Besides fields and methods, a Java class can also contain other classes.
- The rules for using such classes are similar to fields and methods.
  - Access specifier:
    - Just like any other class, a class defined inside of another class can be public, protected, package private, or private.
  - Scope specifier:
    - Just like any other class, a class defined inside of another class can be static or non-static.
      - When it is defined as static, it is called a *nested* class.
      - When it is non-static, it is called an *inner* class.
      - The enclosing class is called the *outer* class.
      - The members (i.e. fields, methods, classes) of a static (nested) class can access to only static members of the outer class.
      - The members of an inner class can access ALL members of the outer class.

Nested and Inner Classes (cont.)

- Usage:
  - Nested classes are used mostly to avoid name clash and to promote information hiding.
  - Inner classes are used to create objects that have direct access to the internals of the outer object and perform complex tasks that simple methods cannot do.
    * An inner object can be thought as an extension of the outer object.
    * Event listeners for Java GUI components are implemented as inner classes.
    * In the state design pattern, the states of an object are often implemented as inner objects. Since an inner object has access to its outer object (the context), there is no need to have setter and getter methods for the state.

Example

- In the hangman game, a character in the target word can be either in the hidden state or visible state.
  - When it is hidden it converts to a String as "_".
  - When it is visible, it converts to a String as the String consisting of its actual character value.

Example (cont.)

- We can apply the state pattern here to implement hangman characters as objects with states. The pattern calls for the following design steps:

  1. Define a WordChar class to represent the characters in a hangman word.
  2. Define an abstract AState class as a private static nested class of WordChar.
  3. Define inner classes HiddenState and VisibleState of WordChar as concrete subclasses of AState. AState and its concrete variants represent the states of a WordChar.
  4. Define a field in WordChar to reference an AState, its current state. All method calls in WordChar are delegated to its state.

Example (cont.)

- The UML diagram on the handout illustrates the above design.
- This design makes use of the composite pattern, the state pattern, and the singleton pattern.
  - By implementing the hangman word as a system of cooperating objects in this manner, you will gain a better understanding of the object-oriented programming design and concepts.