Chapter 1
Singleton Design Pattern

Concepts and Techniques

H.S. Sarjoughian

CSE 460: Software Analysis and Design

School of Computing, Informatics and Decision Systems Engineering
Fulton Schools of Engineering

Arizona State University, Tempe, AZ, USA

Design Patterns

A Design Pattern offers a generic solution to a recurring problem from which for a specific problem, a specialized solution can be derived.

> "A Design Pattern provides a scheme for **refining** the subsystems or components of a software system, or the relationships between them. It describes a **commonly-recurring structure** of communicating components that solves a **general design problem** within **a particular context**" [GoF, 1995]

A design pattern *implicitly promises* that (1) it can satisfy customer's needs and (2) the solution is feasible.

A Conceptual Roadmap to Software Architecture

Software Architecture Model

Design Pattern Models

typing | concurrency | persistence

abstraction | **hierarchy**

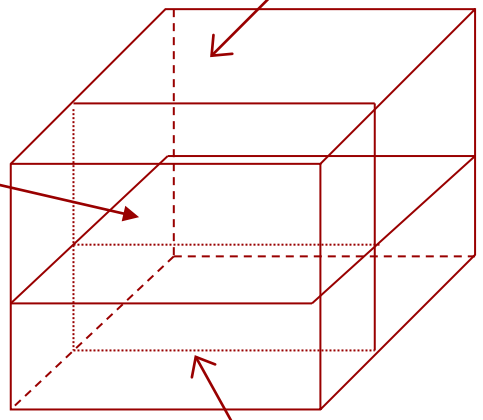
encapsulation | **modularity**

Object Model

UML Model

Logical/Dynamic:
interaction, … diagrams

Logical/Static:
Class, … diagrams



Physical/Static:
Component, … diagrams

Patterns

- Each Design Pattern presents a concrete solution schema for recurring design problems based on proven solutions
  - *Problem requirements* and *desired properties of a solution* are available
- Each Design Pattern provides concepts and specifications distinct from, but complementary to, those contained in the Object Model, UML, and Software Architectures (often tied to particular programming languages or frameworks)
  - Accounts for quality attributes
- Design Patterns document *what* models to develop and *how to* create them
  - Usually in terms of class, sequence, and other UML diagrams (e.g., see the Observer pattern)

Design Pattern Space

		Purpose		
		Creational	**Structural**	**Behavioral**
Scope	**Class**	▪ Factory Method	▪ Adapter (class)	▪ Interpreter ▪ Template Method
	Object	▪ Abstract Factory ▪ Builder ▪ Prototype ▪ ***Singleton***	▪ Adapter (object) ▪ Bridge ▪ Composite ▪ Decorator ▪ ***Façade*** ▪ Flyweight ▪ Proxy	▪ Chain of responsibility ▪ Command ▪ Iterator ▪ Mediator ▪ Memento ▪ ***Observer*** ▪ State ▪ Strategy ▪ Visitor

source: GoF, 1994

Singleton: A Simple Design Pattern

- This design pattern ensures that **only a single instance** of a class (singleton) can exist and there exists **a global point of access to it**
  - windows manager, print spooler, access to a database are examples were the singleton pattern is appropriate
- This design pattern plays an important role when there is a requirement for one instance of a class to exist and it is accessible to clients from a well-known access point
  - may or may not allow a client programmer to create an instance of the class

A design pattern can be described in terms of its **intent**, **motivation**, **applicability**, **structure**, **participants**, **collaborations**, **consequences**, **implementation**, **known uses**, and **related patterns**

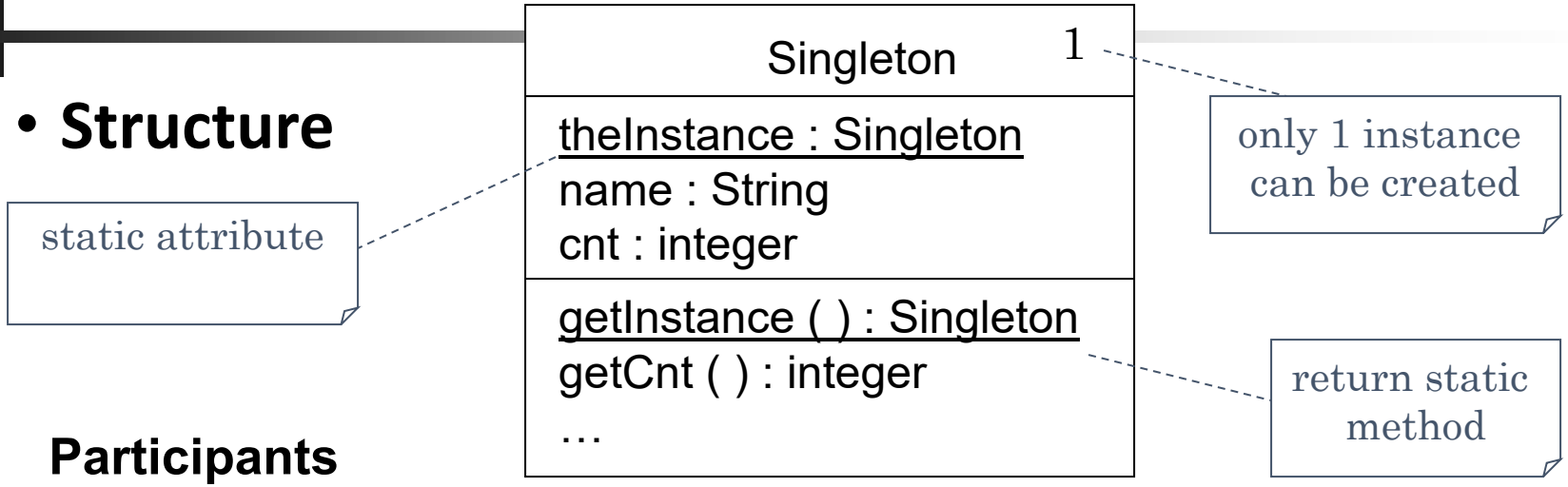
Describing a Pattern: Singleton

Singleton pattern can be described in terms of

- **Intent**
  - ensure a class only has one instance, and provide a global point of access to it
- **Motivation**
  - avoids creating a global variable and making it possible for others to create multiple objects
  - makes the singleton class responsible for creating only one instance of itself
- **Applicability**
  - there must exist only one instance of a class and accessible to clients from a well-known access point

Describing a Pattern: Singleton (Cont.)

- **Structure**



- **Participants**
  - Singleton
    - Defines a class operation (e.g., static in Java) that allows clients to access its unique instance.
    - Maybe responsible for creating its own unique instance; a client may also create the instance

- **Collaborations**
  - clients access a Singleton instance solely through Singleton's class operation.

Describing a Pattern: Singleton (Cont.)

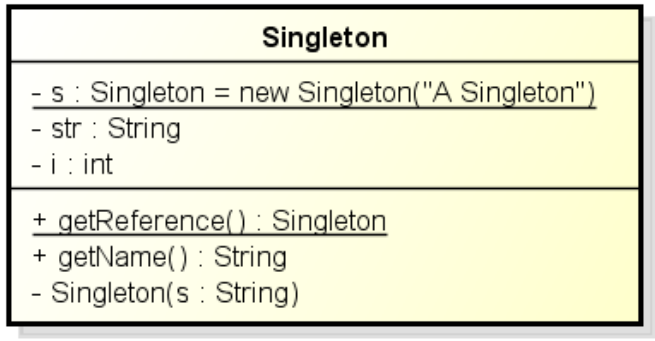
Implementation

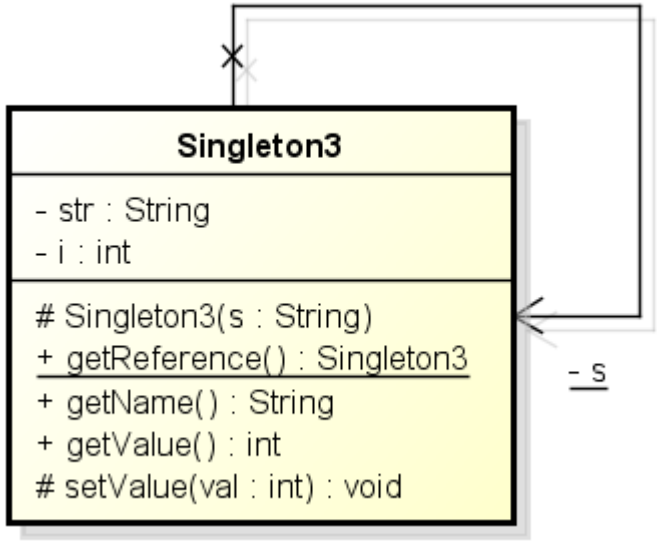
```
final class Singleton {

    //private
    private static Singleton s = new Singleton ("A Singleton");
    private String str;
    private int i;

    private Singleton (String s) {str = “s”;}

    //public
    public static Singleton getReference( ) {return s;}
    public String getName( ) {return str;}
    public int getValue( ) {return i;}
    public void setValue(int val) {i=val;}

}
```

class (static) operation

Astah – Forward Engineering Singleton Class

```
public final class Singleton3 {

  private String str;
  private int i;
  private static Singleton3 s =
    new Singleton3("A Singleton");

  protected Singleton3(String s) {}

  public static Singleton3 getReference() {
      return null;}

  public String getName() {
      return null;}

  public int getValue() {
      return 0;}

  public void setValue(int val) {}

}
```





Describing a Pattern: Singleton (Cont.)

Implementation

```
public class SingletonPattern {

  public static void main(String[ ] args) {

    Singleton s1 = Singleton.getReference( );
    System.out.println(s1.getValue( ));
    System.out.println(s1.getName( ));

    Singleton s2 = Singleton.getReference( );
    s2.setValue(17);
    System.out.println(s1.getValue( ));

  }
}
```

Describing a Pattern: Singleton (Cont.)

- **Consequences (Benefits)**
  - Controlled access to a single, unique, instance
  - Support refinement of operations and representation
  - Permit a variable number of instances
  - Reduced name space
- **Related patterns**
  - Abstract Factory
  - Builder
  - Prototype

Classification of Design Patterns

Creational

- Purpose: handle creation of objects – separate the details of object creation and thus help keeping changes local to the objects (e.g., Singleton)

Structural

- Purpose: support design of objects to satisfy particular project constraints – objects are connected in a such a way that changes in the structure does not require changes in the connections (e.g., Façade)

Behavioral

- Purpose: support objects to handle specific types of actions – encapsulate details of processes (e.g., Observer)

How to Use a Design Pattern

- Read the pattern once through for an overview – Applicability and Consequences are important
- Study the Structure, Participants, and Collaborations sections
- Study the sample code to understand choices from going from design to implementation
- Choose names for pattern participants that are meaningful in for the application at hand (take into account context of the problem)

How to Use a Design Pattern

- Define the classes including interfaces and other classifiers
- Define application-specific names for operations in the pattern
- Implement the operations to carry out the responsibilities and collaborations in the pattern

Selecting a Design Pattern

- Study how design patterns can solve design problems (design patterns help identify suitable objects that have right level of granularity and specify object interfaces
- Understand design patterns Intent sections
- Study how patterns interrelated
- Understand patterns that have similar purpose
- Examine causes of redesign
- Determine what should be considered as variable in your design

Summary

- Design Patterns can provide quick help in solving many design problems – a design pattern support one or more software quality attributes (e.g., modifiability and performance)
- A design pattern offers suitable level of abstractions (e.g., choice of objects and their interactions)
- Design patterns complement software architecture design – some levels of details are not suitable for consideration in software architecture design
- There may not necessarily exist any single perfect design pattern
- Design patterns may be necessary in order to solve multiple problems often faced in large-scale designs (different design patterns solve different quality attributes)

References

- *Design Patterns: Elements of Reuseable Object-Oriented Software, E. Gamma, R. Helm, R. Johnson, J. Vlissides, Addison Wesley, 1994. [commonly referred to as GoF]*
- *Pattern-Oriented Software Architecture, Volume 4, A Pattern Language for Distributed Computing, F. Buschmann, K. Henney, D. C. Schmidt, Wiley, 2007*
- *Pattern-Oriented Software Architecture: A System of Patterns, F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, M. Stal, John Wiley & Sons, 1996.*
- *Thinking in Patterns with Java, B. Eckel, Mindview.net, 2002.*
- *astah, UML software tool, astah.net, 2021*