CMSC 132: OBJECT-ORIENTED PROGRAMMING II

Singleton and Decorator Design Patterns

Department of Computer Science
University of Maryland, College Park

Singleton Pattern

- **Typical problem: only one instance of a class is allowed/needed**
- **Definition**
  - One instance of a class or a value accessible globally
  - In some scenarios only one instance of a class is allowed as more than one instance will be incorrect. For example, you can only have one database manager (several will create data inconsistencies, only one president, etc.)
- **How to define a class as a singleton**
  - Define the class constructor private to control creation of instances
  - Only instance created by static method or when class is loaded
  - Access to single instance only via specific methods (e.g., static method that returns reference to single object reference)
  - Define class as final

Singleton Pattern

- **Examples:**

```
public class Employee {
    public static final int ID = 1234;      // ID is a singleton
}

public final class MySingleton {

    // Declares the unique instance of the class created when class is loaded
    private static MySingleton uniq = new MySingleton();

    // private constructor only accessed from this class
    private MySingleton() { … }

    // Returns reference to unique instance of class
    public static MySingleton getInstance() {
        return uniq;
    }
}
```

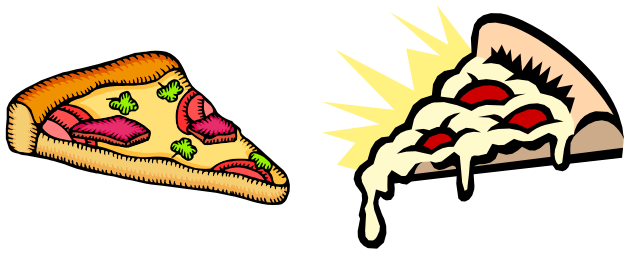
- **Example:** DatabaseManager.java
- How can you modify the singleton design patterns so you only allow only a particular number of objects to be created?

Decorator Pattern

- **Typical problem: customization**
  - A Pizza can have different toppings; defining a class for each possible topping's combination could lead to a large number of subclasses
  - A car in a dealership can have different options (e.g., customized radio, wheels, guarantees, etc.) you can add before buying
- **Definition**
  - Attach additional responsibilities or functions to an object dynamically or statically
- **Where to use & benefits**
  - Provide flexible alternative to subclassing
  - Add new function to an object without affecting other objects
  - Make responsibilities easily added and removed dynamically & transparently to the object

Decorator Pattern

- **Example**
  - Pizza decorator adds toppings to Pizza
  - **Original**
    - Pizza subclasses
    - Combinatorial explosion in # of subclasses
  - **Using pattern**
    - Pizza decorator classes add toppings to Pizza objects dynamically
    - Can create different combinations of toppings without modifying Pizza class
    - **Example:** PizzaDecoratorCode



Decorator Pattern

- **Examples from Java I/O**
  - Interface
    - InputStream
  - Concrete subclasses
    - FileInputStream, ByteArrayInputStream
  - Decorators
    - BufferedInputStream, DataInputStream
  - Code
    - InputStream s = new DataInputStream(new BufferedInputStream (new FileInputStream()));