SINGLETON
pattern

Artur Juvé Vidal
Enginyeria del Software I
EINF 2006-2007

Definicions (1/1)

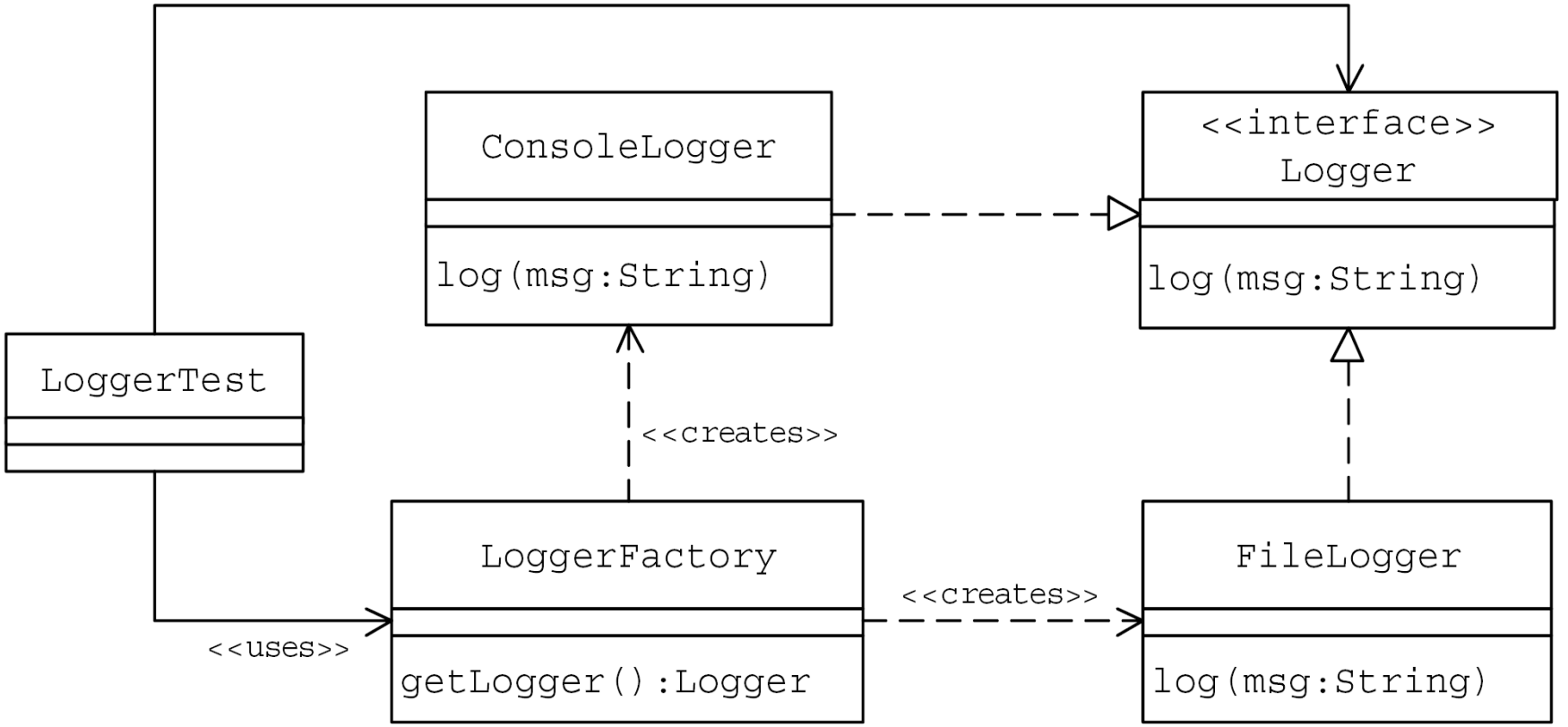
- El patró SINGLETON és un tipus de patró creador.
- S'utilitza quan es necessita (o bé és suficient), que només hi hagi una única instància d'una classe en tota una aplicació.
  - Ex. objecte per a la connexió amb una base de dades.
- La classe responsable de controlar el seu nombre d'instàncies és anomenada classe Singleton.

Exemple (1/3)

- Aquest exemple és continuació de l'exemple vist en el patró factoria.
- L'exemple tracta del disseny d'un sistema automàtic per mostrar missatges.
- Es poden mostrar els missatges a pantalla o bé en un fitxer de text.
- Es crea una classe interface anomenada Logger i dues subclasses anomenades FileLogger i ConsoleLogger.

Exemple (2/3)

- Disseny del sistema de missatges aplicant el patró factoria.

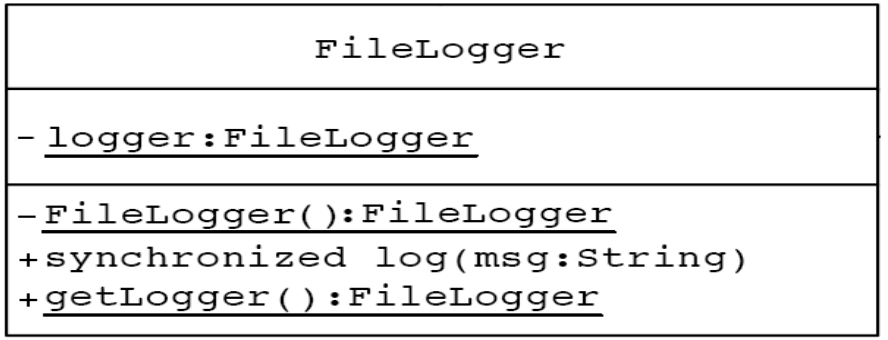


Exemple (3/3)

- L'exemple es centra en la classe FileLogger.
- Interessa aplicar el patró singleton a la classe FileLogger perquè només hi ha un únic fitxer físic a on escriure els missatges.
- Primera solució: Crear una variable global amb una instància de la classe.
  - No preveu que els clients no puguin realitzar noves instàncies de les classes.
  - No preveu que múltiples fils d'execució utilitzin els mètodes de la variable global.

Solució (1/3)

- Es proposa la següent solució:




```
public class FileLogger implements Logger {
  private static FileLogger logger;
  //Prevent clients from using the constructor
  private FileLogger() {
  }
  public static FileLogger getFileLogger() {
    if (logger == null) {
      logger = new FileLogger();
    }
    return logger;
  }
  public synchronized void log(String msg) {
    FileUtil futil = new FileUtil();
    futil.writeToFile("log.txt",msg, true, true);
  }
}
```

Solució (2/3)

- L'atribut estàtic globalitza l'única instància de la classe. Es defineix privat per evitar l'accés directe sense control ja que l'objecte podria no existir.
- El constructor privat evita que els clients creïn noves instàncies de la classe sense controlar si ja existeix una instància prèvia.
- El mètode estàtic getFileLogger() és accessible per qualsevol client. Aquest mètode retorna la instància de la classe creada. Si no existeix el mètode de classe invoca al constructor.

Solució (3/3)

- El mètode getLogger es declara synchronized per evitar possibles accessos a fitxer de més d'un client alhora.
- El mètode getLogger() de la classe LoggerFactory no pot invocar el constructor llavors ha d'utilitzar el mètode estàtic getFileLogger() de la classe FileLogger.

```
public Logger getLogger() {
  if (isFileLoggingEnabled()) {
    return FileLogger.getFileLogger();
  } else {
    return new ConsoleLogger();
  }
}
```

Annexe (1/2)

- Exemples d'implementacions del patró Singleton en Java (http://es.wikipedia.org/)

Una implementación **correcta** en el lenguaje de programación Java para programas multi-hilo es la solución conocida como "inicialización en demanda" sugerida por Bill Pugh:

```
public class Singleton {
    // El constructor privado no permite que se genere un constructor por defecto
    // (público)
    private Singleton() {}

    public static Singleton getInstance() {
        return SingletonHolder.instance;
    }

    private static class SingletonHolder {
            private static Singleton instance = new Singleton();
    }
}
```

Annexe (2/2)

A continuación, un ejemplo **correcto** de inicialización diferida. Se deja para comentar un error común en Java al no tener en cuenta la sincronización de métodos.

```
public class Singleton {
    private static Singleton INSTANCE = null;

    // Private constructor suppresses
    private Singleton() {}

    // creador sincronizado para protegerse de posibles problemas  multi-hilo
    // otra prueba para evitar instantiación múltiple
    private synchronized static void createInstance() {
        if (INSTANCE == null) {
            INSTANCE = new Singleton();
        }
    }

    public static Singleton getInstance() {
        if (INSTANCE == null) createInstance();
        return INSTANCE;
    }
}
```