



תבניות תיכון (Design Patterns)

תבניות תיכון - מוטיבציה

- בחיי היום יום אנחנו מתארים דברים תוך שימוש בתבניות חוזרות:
 - "מכונת א' היא כמו מכונת ב', אבל יש לה 2 דלתות במקום 4"
 - "אני רוצה ארון כמו זה, אבל עם דלתות במקום מגרות"
- גם בפיתוח תוכנה, אנחנו יכולים להסביר כיצד לעשות משהו ע"י התייחסות לדברים שעשינו בעבר, ובצורה כזאת להקל על התקשורת עם עמיתים.
 - "נאחסן את המבנה בעץ בינרי, ונבצע חיפוש לרוחב"

תבנית תיכון - הגדרה

■ **תבנית תיכון:** פיתרון מקובל לבעיית תיכון נפוצה בתכנות מונחה עצמים.
■ התבנית אינה מתארת אלגוריתם ספציפי, אלא קשר בין המחלקות
■ הגדרות מהספרות:

- "Design Patterns are **recurring solutions to design problems** you see over and over." [Alpert, Brown, Wof, 1998].
- "Design Patterns constitute a set of rules describing how to **accomplish certain tasks** in the realm of software development." [Pree, 1994].
- "A pattern address a **recurring design problem** that arises in specific design situations and presents a solution to it." [Buschmann et al., 1996].
- "Patterns identify and specify **abstractions** that are above the level of single classes and instances, or of components." [Gamma et al., 1993].

מקורות

■ המושג נעשה פופולרי בעקבות הספר שמכונה GoF המכיל
דוגמאות קוד ב C++

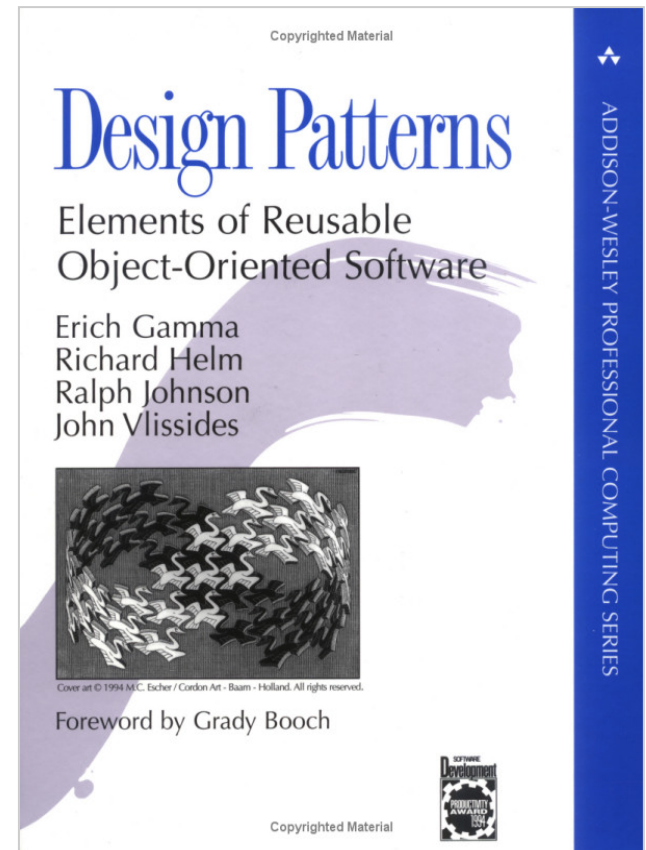
Design Patterns: Elements of Reusable Object-Oriented Software

By Erich Gamma, Richard Helm, Ralph
Johnson, John Vlissides.

Published by Addison Wesley Professional.
Series: Addison-Wesley Professional
Computing Series.

ISBN: 0201633612; Published: Oct 31,
1994; Copyright 1995; Pages: 416;
Edition: 1st.

תוכנה 1 בשפת Java
אוניברסיטת תל אביב



מקורות ב Java

קיימים כמה מקורות לתבניות עיצוב עם דוגמאות בשפת Java כגון: ■

- The Design Patterns Java Companion
James W. Cooper

<http://www.patterndepot.com/put/8/JavaPatterns.htm>

- Thinking in Patterns with Java
Bruce Eckel

<http://www.mindview.net/Books/TIPatterns/>

- Data Structures and Algorithms with Object-Oriented Design
Patterns in Java

<http://www.brpreiss.com/books/opus5/>

תבניות עיצוב רבות אומצו ע"י כותבי הספריות הסטנדרטיות של Java ■

■ בעיקר (אבל לא רק) בספריות GUI

סיווג תבניות

- בספר GoF יש 23 תבניות שמחולקות ל-3 סוגים לפי מטרתן:
 - **תבניות יצירה (Creational):** נוגעות לתהליך היצירה של עצמים.
 - **תבניות מבנה (Structural):** עוסקות בהרכבה של מחלקות ועצמים.
 - **תבניות התנהגות (Behavioral):** מאפיינות את הדרכים בהן מחלקות ועצמים מתקשרים ומחלקים אחריות.
- סיווג אחר: לפי תחום עיסוק התבנית – מחלקות או עצמים.
- בנוסף, ניתן להתייחס **לקבוצות של תבניות** שמופיעים בדרך כלל ביחד:
 - כאלה שמהווים חלופות שונות לפתרון בעיות דומות
 - פתרונות דומים לבעיות שונות
- לתבניות חשיבות גדולה באפיון הבעיות
- חלק מהתבניות ראינו במהלך הקורס – בהקשר רחב או מקומי

סיווג תבניות

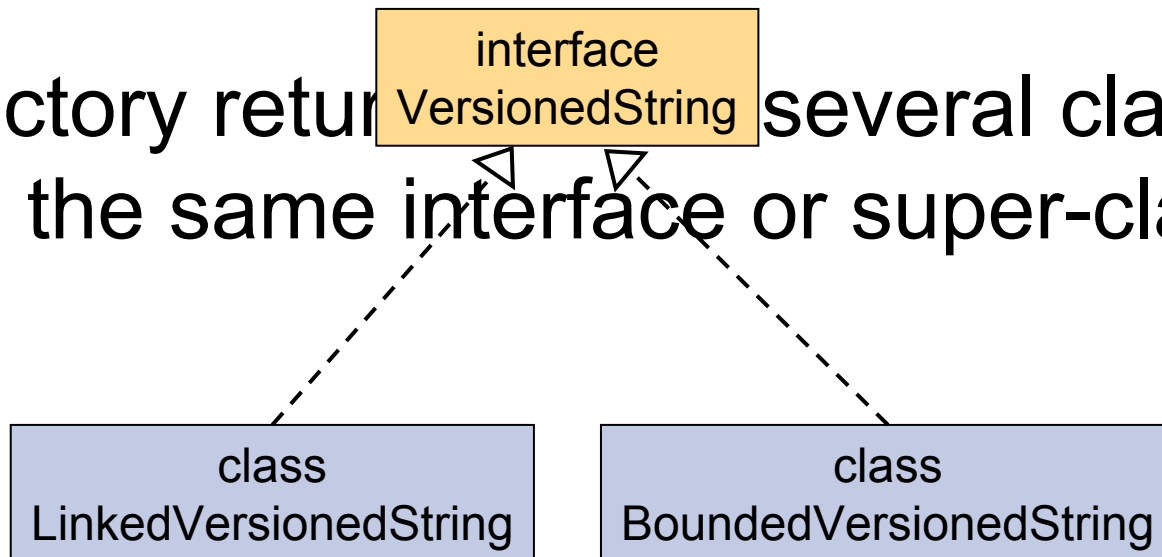
		Purpose		
		Creational	Structural	Behavioral
Scope	Class	Factory Method (107)	Adapter (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Flyweight (195) Observer (293) State (305) Strategy (315) Visitor (331)

Factory

- The `new` operator gets a class name, (not an interface or an abstract class):

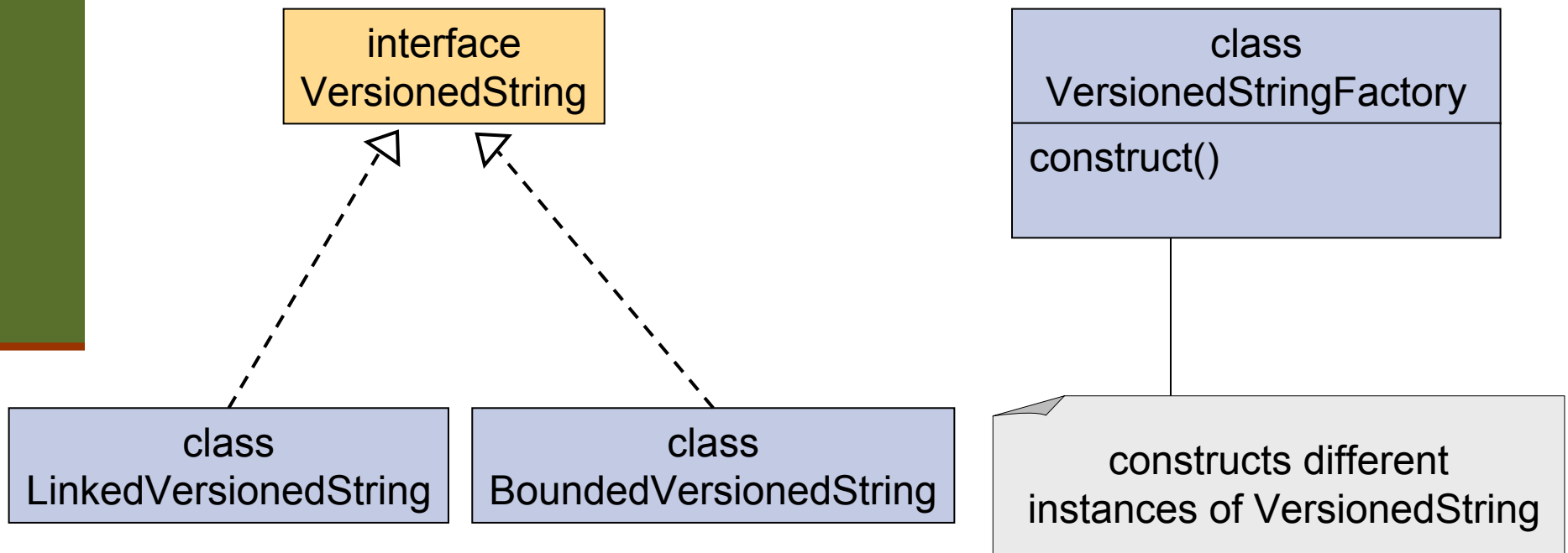
```
VersionedString vstring = new LinkedVersionedString();
```

- A factory returns several classes with the same interface or super-class



Factory (cont.)

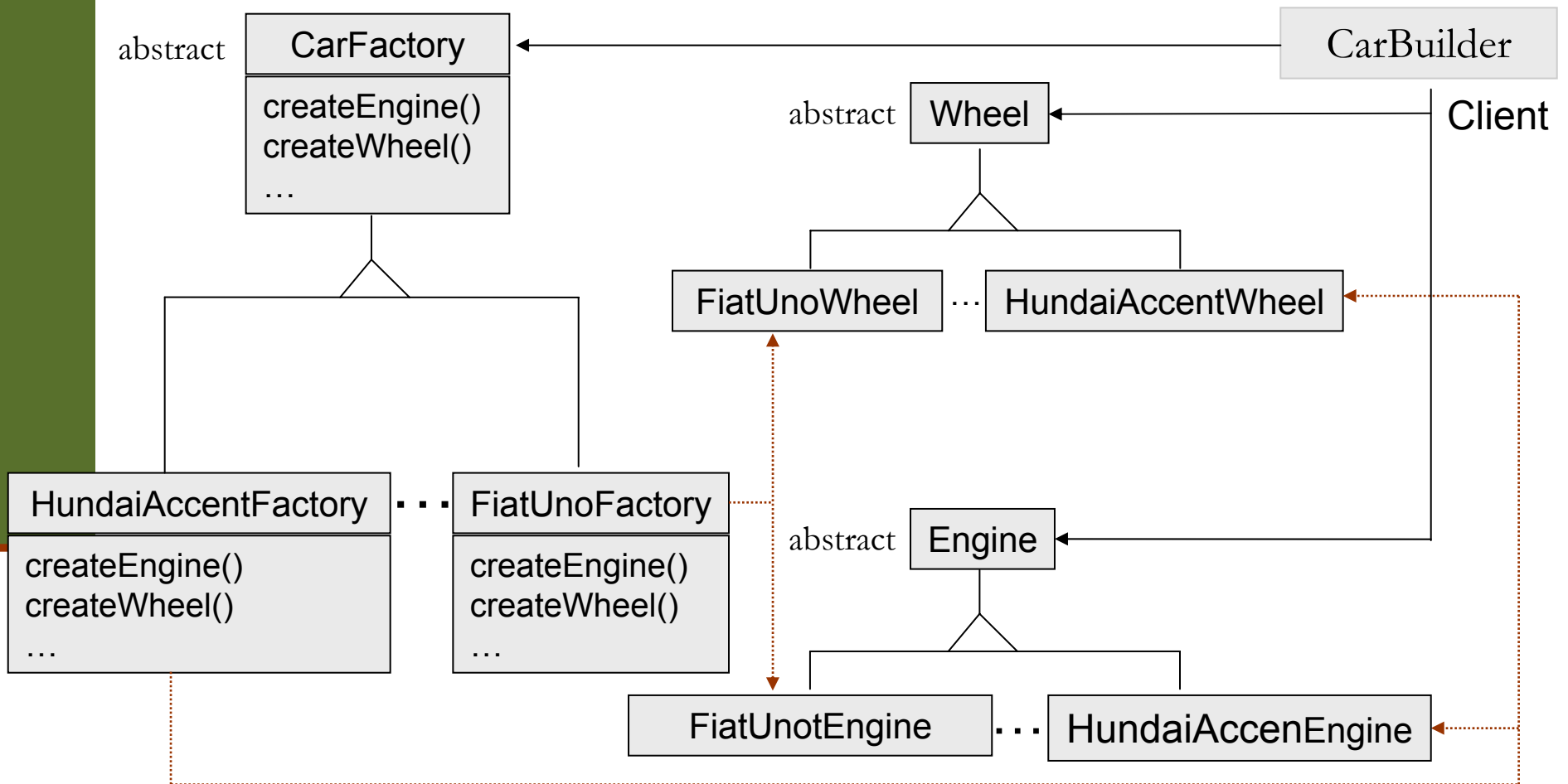
```
VersionedString vstring =  
    VersionedStringFactory.construct();
```



Abstract Factory

- Useful for creating families of related objects without specifying their concrete classes
- Example: An application for building cars
 - builds various types of cars:
Hundai-Accent, Peugeot 205 GTI, Fiat-Uno etc.
 - all cars have the same overall structure, i.e. consist of the same components:
engine, wheels, brakes etc.
 - The components are different.

Abstract Factory (cont.)



Abstract Factory (cont.)

- Isolates concrete classes
- Exchanging product families is easy
- Promotes consistency among products
- Supporting new kinds of products involves changing the `AbstractFactory` class and all of its subclasses.
- Typically implemented as a singleton.

Singleton

- Ensures a class has only one instance and provides a global access point to it.

```
public class Logger {  
    private static final Logger instance = new Logger();  
  
    private Logger() {...}  
  
    public static Logger getInstance() {  
        return instance;  
    }  
}
```

Singleton (cont.)

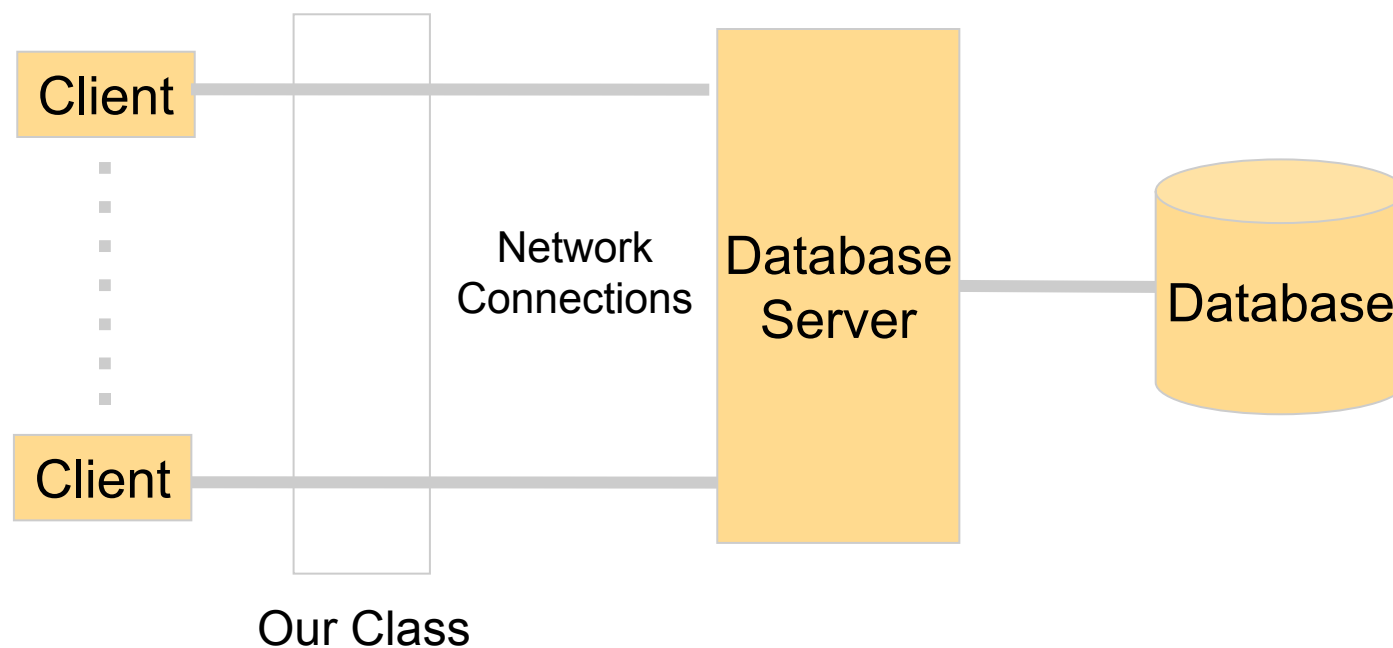
```
public class Logger {  
    private static Logger instance;  
  
    private Logger() {...}  
  
    public static Logger getInstance() {  
        if (instance == null)  
            instance = new Logger();  
  
        return instance;  
    }  
}
```

Lazy evaluation
(not thread-safe)

Object Pool

Database Example:

- Task: Design a class for accessing a DB



Object Pool (cont.)

■ Constraints:

- Establishing and cleaning up connections to a database are time-consuming
- Connecting/Disconnecting time may depend on the number of open connections.
- The number of open connections may be limited (server capacity, DB license)

■ Solution:

- Maintain a pool of open connections for reuse

Object Pool (cont.)

Singleton

```
DBConnectionPool
private DBConnectionPool();
public static DBConnectionPool getInstance();
public int getMaxSize();
public DBConnection acquire();
public release(DBConnection connection);
```



may throw an EmptyPoolException

cannot be constructed by clients.

Reference Objects

- Consider the following case:
 - we have an unlimited pool of DB connections
 - we may end up in an out of memory situation
- To overcome this problem:
 - The pool will use soft references to hold DB connections
 - Unused connections will be cleared by the garbage collector if memory is required.