Intro to Design Patterns and the Unified Modeling Language

CSC207 Fall 2017

Iterator Design Pattern

Context
- A container/collection object.

Problem
- Want a way to iterate over the elements of the container.
- Want to have multiple, independent iterators over the elements of the container.
- Do not want to expose the underlying representation: should not reveal how the elements are stored.

UML

Unified Modeling Language (UML)
A way to draw information about software, including how parts of a program interact.

We’ll use only a small part of the language, Class Diagrams, to represent basic OO design.

Example: Class Person

Data members:
- `name : String []`
- `dob : String`
- `gender : String`

Methods:
- `getName() : String []`
- `setName(name : String [])`
- `getDob() : String`
- `setDob(dob : String)`
- `getGender() : String`
- `setGender(gender : String)`
- `toString() : String`
Notation

Data members:
  name: type

Methods:
  methodName(param1: type1, param2: type2,... ): returnType

Visibility:
  - private
  + public
  # protected
  ~ package

Static: underline

Example: Inheritance

Notation (cont’d)

Abstract method: italic

Abstract class: italic or <<abstract>>

Interface: <<interface>>

Relationship between classes:

  Inheritance

  Interface
**Design Patterns**

A **design pattern** is a general description of the solution to a well-established problem using an arrangement of classes and objects.

Patterns describe the shape of code rather than the details. They're a means of communicating design ideas. They are not specific to any one programming language.

You'll learn about lots of patterns in CSC301 (Introduction to Software Engineering) and CSC302 (Engineering Large Software Systems).

---

**Example: Abstract Class**

```java
<abstract>
    Grade
    + VALID_GRADES: String[] (readOnly)
    + toString(grade: String)
    + gpa(grade: int): double

LetterGrade
- grade: String
+ LetterGrade(grade: String)
+ gpa(grade: double)
+ toString(): String

NumericGrade
- grade: int
+ NumericGrade(grade: int)
+ gpa(grade: double)
+ toString(): String
```

---

**Iterator Design Pattern**

**Context**
- A container/collection object.

**Problem**
- Want a way to iterate over the elements of the container.
- Want to have multiple, independent iterators over the elements of the container.
- Do not want to expose the underlying representation: should not reveal how the elements are stored.

**Iterator Design Pattern: Java**

```java
<interface>
    Iterator<T>
    + hasNext(): boolean
    + next(): T

<interface>
    Iterable<T>
    + iterator(): Iterator<T>

YourIterableClass
- + iterator(): Iterator<T>

Returns an instance of YourIterableClass.
```
Iterator: Example in Java

```
@interface
Iterator<T>
+ hasNext(): boolean
+ next(): T
@end

@interface
Iterable<T>
+ iterator(): Iterator<T>
@end

@interface
AddressBook
+ iterator(): Iterator<Contact>
@end

Returns instance of AddressBookIterator
```