CmpSci 187: Programming with Data Structures
Spring 2015

Lecture #4

John Ridgway

February 5, 2015

1 Abstract Data Types

Abstract Data Types (ADTs)

- Abstraction.
- Information hiding.
- Data abstraction.
- We deal with data at three different levels: application (user/client), logical (abstract), or implementation (concrete).
- Preconditions and postconditions.

Abstract Method

- Only includes return type, name, parameters, and exceptions thrown; *no body.*
- Example:

  ```
  public abstract double getWeight();
  ```

- Only declares the *interface* of the method, not the implementation.

Sample Interface

```
public interface FigureGeometry {
  double PI = 3.14159;

  double getPerimeter();
  double getArea();
  void setScale(double scale);
  double getWeight();
}
```

Sample Implementation

```
public class Circle implements FigureGeometry
{
  private double radius;
  private double scale;

  public Circle(double radius) {
    this.radius = radius; }
  public double getPerimeter() {
    return (2 * PI * radius); }
  public double getArea() {
    return PI * radius * radius; }
  public void setScale(double scale) {
    this.scale = scale; }
  public double getWeight() {
    return scale * this.getArea(); }
}
```

Java Interfaces

- The form of a Java `interface` is:

  ```
  public interface name extends i1, ... {
    declarations
  }
  ```

  where i_j are interfaces that this extends.

- Declarations in an interface can be constants or abstract methods. A constant declaration:

  type variable `=` *expression*`;`

- An abstract method declaration:

  `void` *name*`(`*type name*`, ...) throws` e_1`, ... ;`

  where e_j are exceptions that can be thrown.

Abstraction Clicker Exercise

Which of the following may a Java interface not have?

A. a constructor,

B. an interface extending it,

C. two or more interfaces that it extends, or

D. an abstract method.

The StringLog ADT Specification

- Constructors.
- Transformers: `insert(String element)` and `clear()`.
- Observers: `contains(String element)`, `size()`, `isFull()`, `getName()`, and `toString()`.
- The `StringLog` interface.
- Using the `StringLog` interface.

The StringLogInterface (simplified)

```
package ch02.stringLogs;
public interface StringLogInterface {
  /** Places element into this StringLog.
   * Precondition: This StringLog is not full. */
  void insert(String element);
  /** Returns true if this StringLog is full. */
  boolean isFull();
  /** Returns number of Strings in this StringLog. */
  int size();
  /** Returns true if the element is in this StringLog.
   * Ignores case differences in comparison. */
  boolean contains(String element);
  /** Returns the name of this StringLog */
  String getName();
  /** Returns a nice String representing this StringLog. */
  String toString(); }
```

Observer/Transformer Clicker Exercise?

Suppose I have a class `Locomotive` that describes a locomotive, with methods `Person getDriver()` that returns the locomotive's driver, and `void addFuel(double quantity)` that increases the amount of fuel available. What are these methods?

A. both are observers,

B. both are transformers,

C. `getDriver()` is an observer, `addFuel()` is a constructor,

D. `getDriver()` is an observer, `addFuel()` is a transformer,

E. `getDriver()` is a transformer, `addFuel()` is an observer.

Array-Based StringLog ADT Implementation

- Instance variables: `name`, `log`, and `size`.
- Transformers: `insert(String element)` and `clear()`.
- Observers: `contains(String element)`, `size()`, `isFull()`, `getName()`, and `toString()`.

Boolean Field `full` Clicker Exercise

Suppose we decided to have a boolean field `full` that we could just return when the `isFull()` method is called. Which of these statements would be **false**?

A. `full` might have to be modified by each transformer,

B. the `isFull()` method would run more quickly,

C. `full` would never have to be changed by any observer method,

D. `full` never changes from `true` to `false`.

2 Testing

Software Testing

- The book talks about building tests manually; we'll use JUnit.
- Identifying test cases.
- Test plans.
- Testing ADT implementations.