CS2123 Data Structures
Review Object-Oriented Programming

Weining Zhang
Department of Computer Science
University of Texas at San Antonio
August 28, 2010

Outline

1. OO Programming and Interfaces
   - OO Programming Concepts
   - Interfaces
   - Generic Programming

This Week’s Tasks

- Review concepts of Object Oriented Programming
- Learn Java collection classes and iterators
- Learn inheritance and interface
- Learn incremental design technique

Classes and Objects

- A class is a specification (or blueprint) of a set of objects
  - The class name is also a type name
  - Data fields
  - Methods

- An object is an instantiation of a class
  - At instantiation the object is assigned a section of memory
  - Its data fields are initialized by a constructor
  - The values of the data field can only be accessed using the methods

- An OOP program solves problems by specifying a set of classes and using objects of various classes to accomplish required tasks
In Java, an object can only be accessed via a reference, which is a variable holding the starting address of the object.

```java
public class Book {
    private String title;
    private String authors;
    private String ISBN;
    private double price;
}

Book myBook = new Book("Java Software Structures", "Lewis and Chase", "0-13-607858-6", 60.0);
```

A class can be specified by extending another class.

```java
public class A extends B {...}
```

- Class A is a subclass and class B is a superclass.
- If a class does not extend any class, it is a base class.

A subclass inherits all public and protected fields and methods of all superclasses.

- It can override (have a different version) of the inherited methods.
- It refers its immediate superclass as the `super`.

An interface is a specification of a set of methods. It is intended to define required behaviors of a class.

An interface can be implemented by one or more classes. These classes must specify how each method in the interface is implemented.

```java
public interface Comparable<T> {
    public int compareTo(T o);
}
```

```java
public interface Iterator<E> {
    public boolean hasNext();
    public E next();
    public void remove();
}
```

Java's Set Interface

```java
public interface Set<E> extends Collection<E> {
    public boolean add(E e)
    public boolean addAll(Collection<? extends E> c)
    public void clear()
    public boolean contains(Object o)
    public boolean containsAll(Collection<?> c)
    public boolean equals(Object o)
    public int hashCode()
    public boolean isEmpty()
    public Iterator<E> iterator()
    public boolean remove(Object o)
    public boolean removeAll(Collection<?> c)
    public boolean retainAll(Collection<?> c)
    public int size()
    public Object[] toArray()
    public <T> T[] toArray(T[] a)
}
```
**Case Study**

- **Description**
  A WordReader is an object that, given a text file, returns the next word upon a request.

- **Tasks**
  - Develop a WordReader class to implement the `Iterator<String>` interface
  - Develop a WordReaderTester class to use and test the WordReader class

**Generic Programming**

- It is more convenient to design classes or methods to handle data whose class is not known at the time of programming.
- The unknown type is represented by a type variable (alias or parameter), usually E or T.
- We can define a static generic method, a generic class or a generic interface.

**Example**

```java
public static <T> T findMax(T[] a) {...}
public class ArrayList<T> {...}
public interface List<T> {...}
```

**Motivation**

Consider a method `findMax` that finds the maximum element in an array. Need different methods for arrays containing Integers, Students, and Books.

- `Integer findMax(Integer[] myArray)`
- `Student findMax(Student[] myArray)`
- `Book findMax(Book[] myArray)`

Only need one method if we use generic method

- `T findMax(T[] myArray)`

Type variable T can be instantiated when the method is used.

**Limitations**

- Must use real classes with a generic type.

**Example**

```java
List<Integer> = new LinkedList<Integer>(); // OK
List<int> L = new LinkedList<int>(); // wrong
```

- Cannot instantiate a generic type

**Example**

```java
T obj = new T(); // invalid
```
Cannot create an array of a generic type

Example

```java
T[] a = new T[10]; // invalid
T[] a = (T[]) new Object[10]; // valid
```