CSCI 3230 Data Structures

Introduction

Weitian Tong, Ph.D.

Department of Computer Science
Georgia Southern University
Website: www.weitiantong.com
Email:wtong@georgiasouthern.edu

Table of contents

1. Course content

2. Warming-up

Course content

Program

Programs = Data Structures + Algorithms

Data structure

An organization of information, usually in memory, such as array, queue, or stack.

Algorithm

A finite set of unambiguous instructions performed in a prescribed sequence to achieve a goal, especially a mathematical rule or procedure used to compute a desired result.

What We Will Study

Algorithm analysis: big-Oh notation, asymptotic analysis
Algorithm paradigms: brute-force, divide and conquer, recursion, greedy algorithm, dynamic programming (optional)
Algorithms:

- Basic operations on learned data structures
- Sorting, Searching, Graph algorithms

Data Structures:

- **Linear Data Structures**

  Arrays, ArrayList, Linked Lists, Stacks, Queues

- **Non-Linear Data Structures**

  Trees, Heaps, Hash Tables, Search Trees, Graphs

Please see syllabus posted on website for detailed schedule (tentative).

Warming-up

Design Patterns

Definition:

- A template for a software solution that can be applied to a variety of situations.
- Main elements of solution are described in the abstract.
- Can be specialized to meet specific circumstances.

Example algorithm design patterns:

brute-force, divide and conquer, recursion, greedy

Principles of Object Oriented Design (OOD)

Programs consist of objects.

Objects consist of

- Data structures
- Algorithms to construct, access and modify these structures.

Important concepts:

Abstraction, Encapsulation, Modularity, Hierarchical Organization

More technical details

Other important concepts:

- **Inheritance**: subclass, superclass
- **Method Overriding**:
  - Generally methods of a subclass replace superclass methods.
  - An exception is constructor methods, which do not replace, but refine superclass constructor methods.
- **Polymorphism**
- **Method overroading**

More technical details

Continue:

- Abstract Data Types and Interfaces
  - A set of data values and associated operations that are precisely specified independent of any particular implementation
  - In Java, an ADT
    - can be expressed by an interface.
    - is realized as a complete data structure by a class.
    - is instantiated as an object.
- Casting
- Generics
- Pseudo-Code; Examples

Thank you!

Questions?

Refinement Overriding: Example 1

```java
public class Camera {
    private String cameraMake;
    private String cameraModel;

    Camera(String mk, String mdl) { //constructor
        cameraMake = mk;
        cameraModel = mdl;
    }

    public String make() { return cameraMake; }
    public String model() { return cameraModel; }
}
```

```java
public class DigitalCamera extends Camera{
    private int numPix;
    DigitalCamera(String mk, String mdl, int n) { //refine
        super(mk, mdl);
        numPix = n;
    }
    public int numberOfPixels() {return numPix;} //extend
}
```

```java
DigitalCamera myCam = new DigitalCamera("Nikon", "D90",
    12000000);
```

GO BACK

Refinement Overriding: Example 2

```java
public class Camera {
    private String cameraMake;
    private String cameraModel;

    Camera(String mk, String mdl) { //constructor
        cameraMake = mk;
        cameraModel = mdl;
    }

    public String make() { return cameraMake; }
    public String model() { return cameraModel; }
}
```

```java
public class DigitalCamera extends Camera{
    private int numPix;
    DigitalCamera(String mk, String mdl) { //refine
        super(mk, mdl);
        numPix = 0;
    }
    public int numberOfPixels() {return numPix;} //extend
}
```

```java
DigitalCamera myCam = new DigitalCamera("Nikon", "D90");
```

GO BACK

Replacement Overriding: Example

```
1  public class DigitalCamera extends Camera{
2      private int numPix;
3      DigitalCamera(String mk, String mdl, int n) { //constructor
4          super(mk, mdl);
5          numPix = n;
6      }
7      public int numberOfPixels() { return numPix; }
8      public byte[][][] getDigitalImage() { return takeDigitalPhoto(); }
9  }
```

```
1  public class AutoDigitalCamera extends DigitalCamera{
2     AutoDigitalCamera(String mk, String mdl, int n) { //constructor
3          super(mk, mdl, n);
4     }
5
6     public byte[][][] getDigitalImage() { //replace
7        autoFocus();
8        return takeDigitalPhoto();
9     }
10 }
```

```
1  DigitalCamera myCam = new AutoDigitalCamera("Nikon", "D90",
       12000000);
2  byte[][][] myImage = myCam.getDigitalImage(); // polymorphism
```

GO BACK

ADT Example

```
1  public interface Device {
2     public String make();
3     public String model();
4  }
```

```
1  public class Camera implements Device {
2      private String cameraMake;
3      private String cameraModel;
4      private int numPix;
5
6      Camera(String mk, String mdl, int n) { //constructor
7          cameraMake = mk;
8          cameraModel = mdl;
9          numPix = n;
10     }
11     public String make() { return cameraMake; }
12     public String model() { return cameraModel; }
13     public int numberOfPixels() { return numPix; }
14 }
```

```
1  Camera myCam = new Camera("Nikon", "D90", 12000000);
```

GO BACK

Casting Example

```
DigitalCamera myCam1 =
        new DigitalCamera("Nikon","D90");
DigitalCamera myCam2 =
        new AutoDigitalCamera("Olympus","E30",12000000);
AutoDigitalCamera myCam3 =
        new AutoDigitalCamera("Sony","A550",14000000);
```

```
myCam1 = myCam3; //
myCam3 = myCam1; //
myCam3 = myCam2; //
myCam3 = (AutoDigitalCamera) myCam1;
//
myCam3 = (AutoDigitalCamera) myCam2;
//
```

GO BACK

Casting Example

```
DigitalCamera myCam1 =
        new DigitalCamera("Nikon","D90");
DigitalCamera myCam2 =
        new AutoDigitalCamera("Olympus","E30",12000000);
AutoDigitalCamera myCam3 =
        new AutoDigitalCamera("Sony","A550",14000000);
```

```
myCam1 = myCam3; // widening conversion
myCam3 = myCam1; //
myCam3 = myCam2; //
myCam3 = (AutoDigitalCamera) myCam1;
//
myCam3 = (AutoDigitalCamera) myCam2;
//
```

GO BACK

Casting Example

```
DigitalCamera myCam1 =
        new DigitalCamera("Nikon","D90");
DigitalCamera myCam2 =
        new AutoDigitalCamera("Olympus","E30",12000000);
AutoDigitalCamera myCam3 =
        new AutoDigitalCamera("Sony","A550",14000000);
```

```
myCam1 = myCam3; // widening conversion
myCam3 = myCam1; // compiler error
myCam3 = myCam2; //
myCam3 = (AutoDigitalCamera) myCam1;
//
myCam3 = (AutoDigitalCamera) myCam2;
//
```

GO BACK

Casting Example

```
DigitalCamera myCam1 =
        new DigitalCamera("Nikon","D90");
DigitalCamera myCam2 =
        new AutoDigitalCamera("Olympus","E30",12000000);
AutoDigitalCamera myCam3 =
        new AutoDigitalCamera("Sony","A550",14000000);
```

```
myCam1 = myCam3; // widening conversion
myCam3 = myCam1; // compiler error
myCam3 = myCam2; // compiler error
myCam3 = (AutoDigitalCamera) myCam1;
//
myCam3 = (AutoDigitalCamera) myCam2;
//
```

GO BACK

Casting Example

```
DigitalCamera myCam1 =
        new DigitalCamera("Nikon","D90");
DigitalCamera myCam2 =
        new AutoDigitalCamera("Olympus","E30",12000000);
AutoDigitalCamera myCam3 =
        new AutoDigitalCamera("Sony","A550",14000000);
```

```
myCam1 = myCam3; // widening conversion
myCam3 = myCam1; // compiler error
myCam3 = myCam2; // compiler error
myCam3 = (AutoDigitalCamera) myCam1;
// run-time exception
myCam3 = (AutoDigitalCamera) myCam2;
//
```

GO BACK

Casting Example

```
DigitalCamera myCam1 =
        new DigitalCamera("Nikon","D90");
DigitalCamera myCam2 =
        new AutoDigitalCamera("Olympus","E30",12000000);
AutoDigitalCamera myCam3 =
        new AutoDigitalCamera("Sony","A550",14000000);
```

```
myCam1 = myCam3; // widening conversion
myCam3 = myCam1; // compiler error
myCam3 = myCam2; // compiler error
myCam3 = (AutoDigitalCamera) myCam1;
// run-time exception
myCam3 = (AutoDigitalCamera) myCam2;
// valid narrowing conversion
```

GO BACK

Generics Example

```java
/** Creates a coupling between two objects */
public class Couple<A, B> {
    A obj1;
    B obj2;

    public void set(A o1, B o2) {
        obj1 = o1;
        obj2 = o2;
    }
}
```

```java
Camera myCam1 = new DigitalCamera("Nikon","D90",12000000);
Camera myCam2 = new AutoDigitalCamera("Olympus","E30",12000000);

Couple<Camera,Camera> stereoCamera=new Couple<Camera,Camera>();

stereoCamera.set(myCam1, myCam2);
```

GO BACK