Object Oriented Design

Design

- The goal is to design a *modular* solution, using the techniques of:
  - Decomposition
  - Abstraction
  - Information Hiding

Abstraction and Information Hiding

- Public view of a module
  - Described by its specifications
- Private view of a module
  - Consists of details which should not be described by the specifications
- Principle of information hiding
  - Hide details within a module
  - Ensure that no other module can tamper with these hidden details

Object oriented design

- Object oriented design a powerful *modular* design.
- Produces a collection of objects that interact with each other to solve the problem.
- An object combines data and operations on that data.
  - Data: class variables.
  - Operations: class methods.
- Methods encapsulate actions while objects encapsulate data as well as actions.

How to achieve an O-O design

- One simple way is to produce objects that represent real-life entities.
- For example if you want to design an object-oriented solution for the registry system of your school the following are candidate objects:
  - The school itself
  - Departments
  - Students
  - Faculties
  - Staffs
  - Courses

Object-Oriented design: Clock example

- Suppose you want to design a program that displays a clock on your screen.
- Problem specification: The program maintain a ***digital clock*** that ***displays*** the time in hours and minutes. The ***hour and minute indicator*** are both digital devices ranging from 1 to 12 and 0 to 59. You should be able to ***set*** the time by setting the hour and minute indicators, and the clock should maintain the time by ***updating*** these indicators.
- We can immediately extract the objects form the above specification.
  - Clock
    - Set the time.
    - Advance the time.
    - Display the time.
  - Hour indicator
    - Set its value
    - Advance its value
    - Display its value
  - Minute indicator
    - Set its value
    - Advance its value
    - Display its value

- The hour indicator and minute indicator are quite similar.
- In fact they are two objects of the same type called ***class***.
- Therefore what we need to specify is not a particular object but a ***class of objects***.
- An object is an ***instance*** of a class.

Class vs. Object in programs.

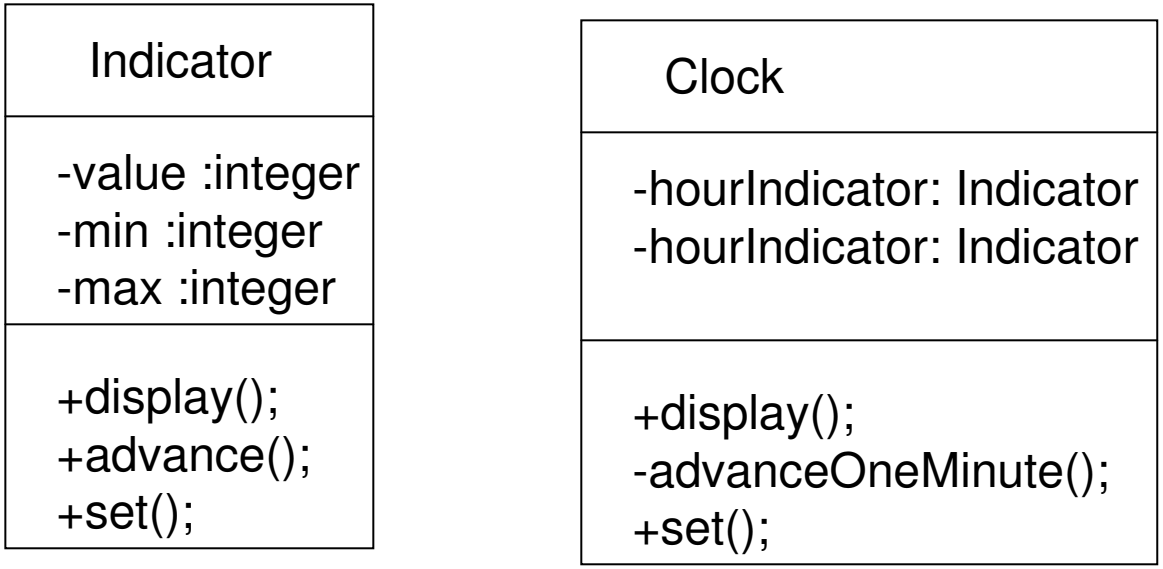
- When you declare a class in your program you tell the compiler that there will objects of this type later. At the moment no actual object is created.

```
public class aClass{
    public int aMember;
    public void aMethod(){
        ...
    }
}
```

- An object is created when you *instantiate* it in somewhere in your program.

```
static public void main(){
    ....
    aClass anObject = new aClass(); //an object is created here.
}
```

Indicator and Clock classes



+ means the member is public.
- means the member is private.

The Indicator Class

```
public class Indicator{
    private int value, min, max;
    public Indicator(int min, int max, int value){
        this.min=min;
        this.max=max;
        this.value=value;
    }
    public Indicator(int min, int max){
        this.min=min;
        this.max=max;
        value=0;
    }
    public void set(int value){
        this.value=value;
    }
    public int  getValue(){
        return value;
    }
    public int display(){
        //print the value at a fixed location on the screen.
    }
    public void advance(){
        value = (value+1) % (max-min+1)+ min;
    }
}
```

```
public class Clock{

    private Indicator hourIndicator, minuteIndicator; //clock has two indicators.

    public Clock(int hour, int minute){
        hourIndicator = new Indicator(1, 12, hour);
        minuteIndicator = new Indicator(0, 59, minute);
    }
    public Clock(){
        hourIndicator = new Indicator(1, 12, 1);
        minuteIndicator = new Indicator(0, 59, 0);
    }
    public void set(int hour, int minute){  //implementing the set operation
        hourIndicator.set(hour);
        minuteIndicator.set(minute);
    }

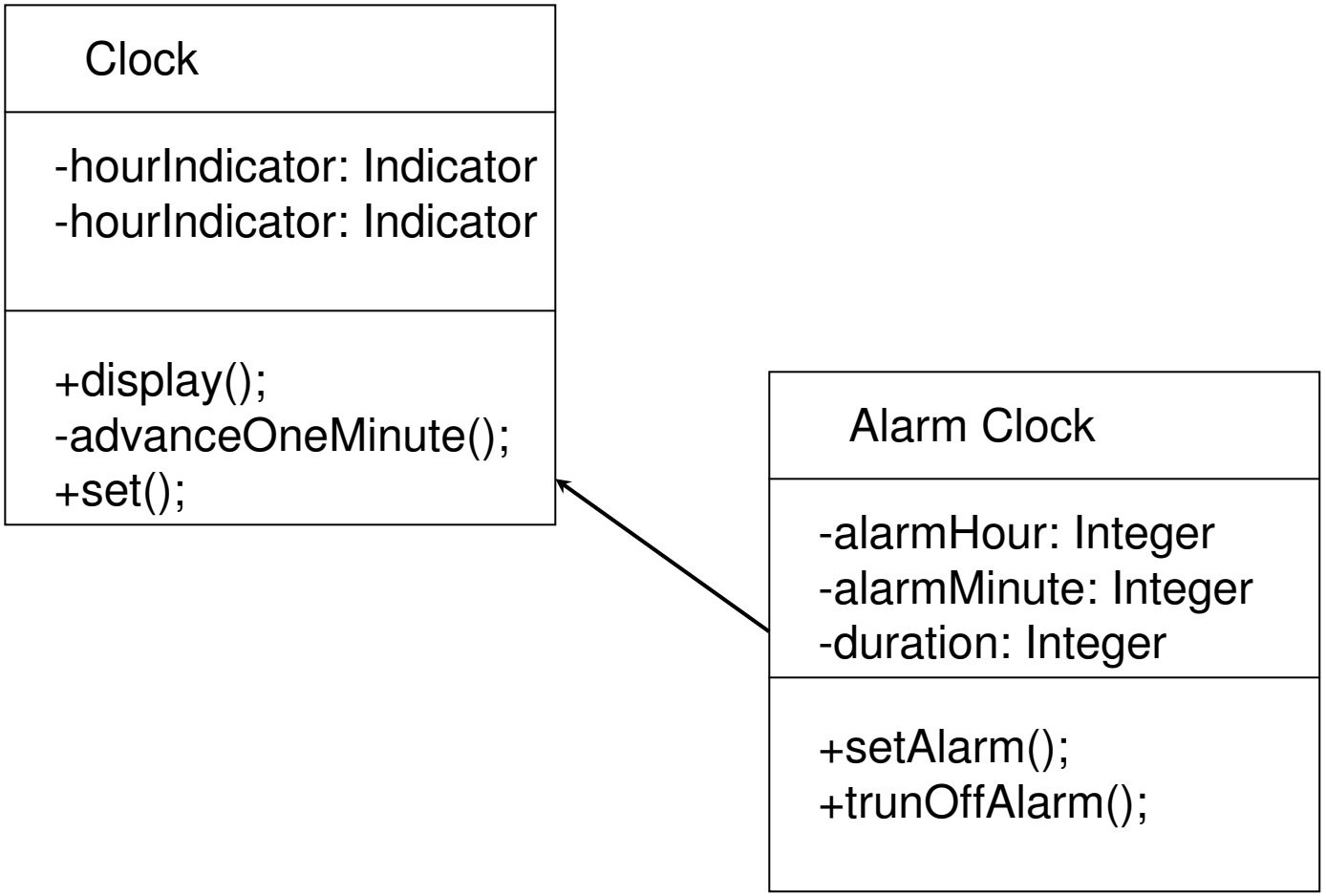
    private void advanceOneMinute(){    //implementing the advance operation
        minuteIndicator.advance();
        if(minuteIndicator.getValue() == 0){
            hourIndicator.advance();
        }
    }

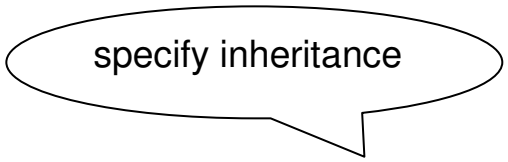
    public void display(){  //implementing the display operation
        while(no stop request){
            hourIndicator.display();
            minuteIndicator.display();
            advanceOneMinute();
            wait one minute;
        }
    }
}
```

```
static public void main(){
    ....
    Clock myClock = new Clock(1, 0);
    myClock.display();
}
```

Inheritance

- Suppose you want to develop an alarm clock.
- An alarm clock has the same functionality as a clock plus it adds operations to provide an alarm.
- If we could somehow *re-use* the design from the clock class, life would be a lot easier.
- Fortunately we can thanks to the *inheritance* tool provided in object-oriented programming languages.


Clock
-hourIndicator: Indicator
-hourIndicator: Indicator
+display();
-advanceOneMinute();
+set();
Alarm Clock
-alarmHour: Integer
-alarmMinute: Integer
-duration: Integer
+setAlarm();
+trunOffAlarm();




```
public class AlarmClock extends Clock{
    private int alarmHour, alarmMinute, duration;
    public void setAlarm(int hour, int minute){

    }
    public void trunOffAlarm(){

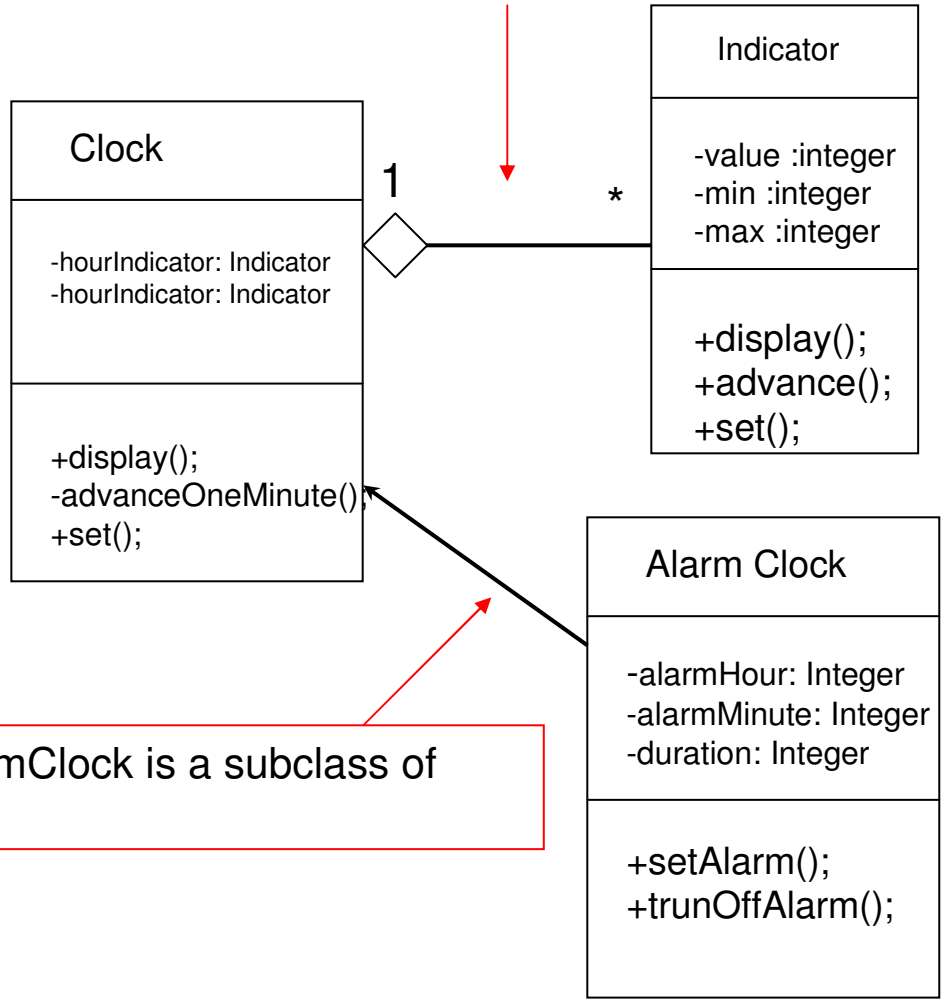
    }
}


static public void main(){
    ....
    Clock myClock=new AlarmClock(1, 0);
    myClock.setAlarm(8, 0); //setAlarm is a mehod defined in the sub class
    myClock.display();  //display method is inherited from the super class.
}
```

Modeling Object-Oriented Designs Using UML

- Unified Modeling Language (UML): language to express OO designs
- Class diagrams include name, data, operations
- Text-based notation: more complete specifications
- Associations between classes are represented by lines.

means a clock can have more than one indicator but an indicator belongs to a single clock.



means AlarmClock is a subclass of Clock

Functional Decomposition

- Object-oriented design (OOD)
  - Produces a data-oriented solution-doesn't say much about the methods.
  - Identifies objects by focusing on the nouns in the problem statement
  - Sometimes the methods within classes are too complicated to be handled at once.
- Functional Decomposition (FD)
  - Produces modular solutions for problems in which the emphasis is on the algorithms
  - Identifies actions by focusing on the verbs in the problem statement
  - A task is addressed at successively lower levels of detail

Functional Decomposition

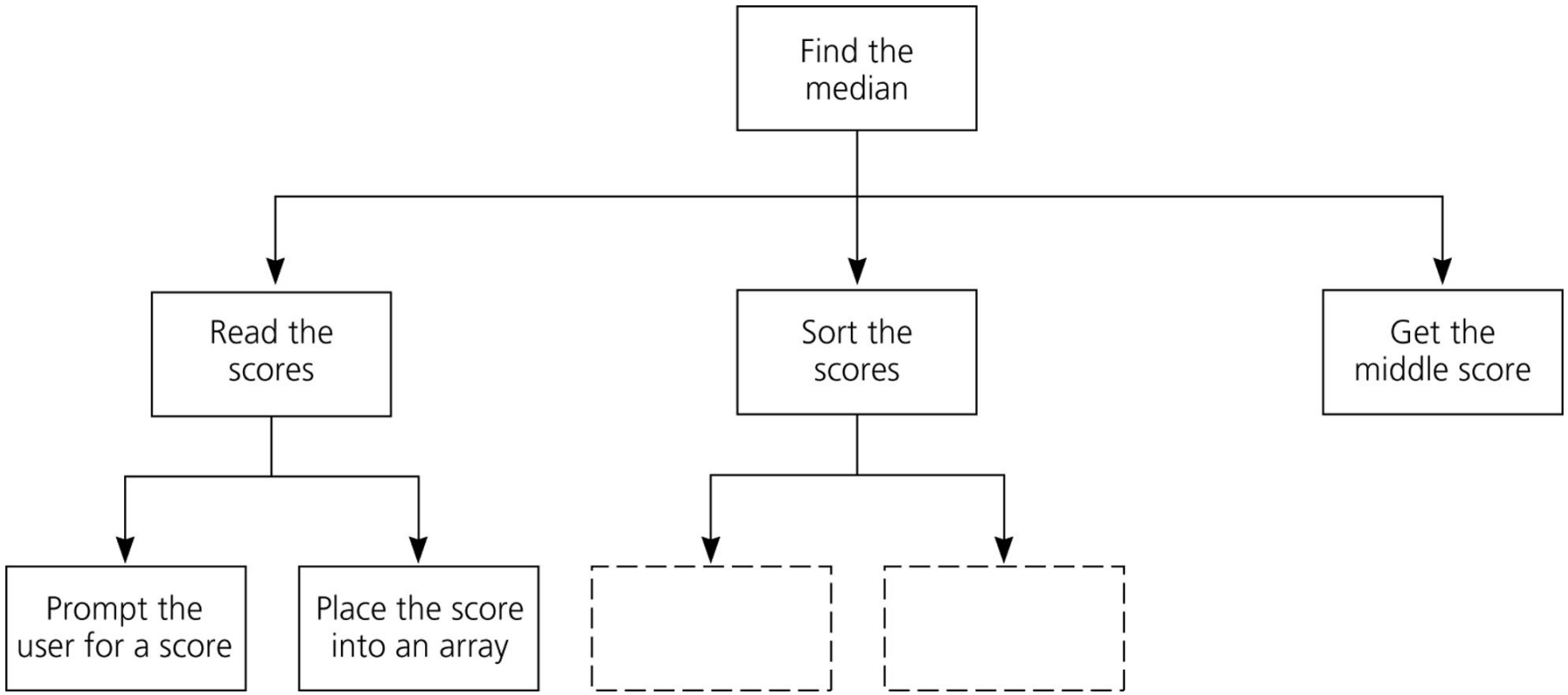


Figure 2-4

A structure chart showing the hierarchy of modules

General Design Guidelines

- Use OOD and FD together
- Use OOD for problems that primarily involve data
- Use FD to design algorithms for an object's operations
- Consider FD to design solutions to problems that emphasize algorithms over data
- Focus on what, not how, when designing both ADTs and algorithms
- Consider incorporating previously written software components into your design