Part 4. Advanced Java Topics

Instructor: Sangmi Pallickara (sangmi@cs.colostate.edu)
Department of Computer Science
Colorado State University

Outline

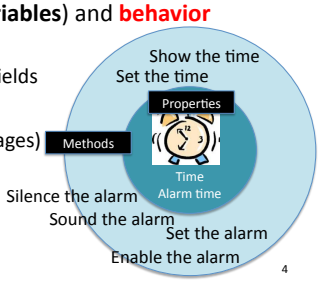
- **Object Oriented Programming**
- Data Encapsulation
- Inheritance
- Polymorphism
- Using Abstract and Interface

Object Oriented Programming

- Programming paradigm using "Objects" : data structures consisting of data fields and methods together with their interaction.
- Object?
- Class?
- Interface?
- Package?

Basic Components: Object

- a software bundle of **related states** (**properties**, or **variables**) and **behavior** (**method**)
  - State is stored in fields (variables in some programming languages)
  - Method exposes object's behavior.



Basic Components

- **Class**: ***Blueprint*** from which objects are created
  - Multiple Instances created from a class
- **Interface**: A ***Contract*** between classes and the outside the world.
  - When a class implements an interface, it **promises to provide the behavior** published by that interface.

Basic Components

- **Package**: a ***namespace*** for organizing classes and interfaces

Outline

- Object Oriented Programming
- **Data Encapsulation**
- Inheritance
- Polymorphism
- Using Abstract and Interface

7

Data Encapsulation

- An ability of an object **to be a container** (or capsule) for related properties and methods.
  - Preventing unexpected change or reuse of the content
- **Data hiding**
  - Object can shield variables from external access.
    - Private variables
    - Public **accessor** and **mutator** methods

8

Data Encapsulation

```
public class Clock
{
  private long time, alarm_time;
  private String serialNo;

  public void setTime(long _time){
      time = _time;
  }
  public void setAlarmTime(long_time){
      alarm_time = _time;
  }
  public long getTime(){return time}
  public long getAlarmTime(){return alarm_time}
  public void noticeAlarm(){ ring alarm }
  protected void set serialNo(String _serialNo){…}
}
```

9

Outline

- Object Oriented Programming
- Data Encapsulation
- ***Inheritance***
- Polymorphism
- Using Abstract and Interface

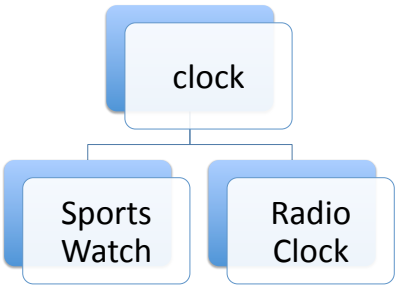
10

Inheritance

- The ability of a class to ***derive*** properties from a previously defined class.
- **Relationship** among classes.
- Enables to **reuse** software components
  - E.g. java.lang.Object()
  - toString(), notifyAll(), equals(), etc.

11

Example: Inheritance



12

Example: Inheritance – cont.

```
Public class SportsWatch extends Clock
{
   private long start_time;
   private long end_time;

   public long getDuration()
   {
     return end_time - start_time;
   }

}
```

13

Overriding Methods

```
public class RadioClock
{
  @override
    public void noticeAlarm(){
          ring alarm
          turn_on_the_Radio
    }
}
```

14

Java Access Modifiers

- Keywords: `public`, `private`, and `protected`
- Control the visibility of the members of a class
  - ***Public members*** can be used by anyone
  - Members declared without an access modifier are available to methods of the class and methods of other classes in the same package
  - ***Private members*** can be used only by methods of the class
  - ***Protected members*** can be used only by *methods of the class, methods of other classes in the same package, and methods of the subclasses*.

15

Outline

- Object Oriented Programming
- Data Encapsulation
- Inheritance
- ***Polymorphism***
- Using Abstract and Interface

16

Polymorphism

- "Having multiple forms"
- Ability to create a variable, or an object that has more than one form.

17

Polymorphic method

```
RadioClock myRadioClock = new RadioClock();
Clock myClock = myRadioClock;
myClock.notifyAlarm();
```

18

Dynamic Binding

- `myClock` actually references an instance of `RadioClock`
  - It will turn on the radio.
- The version of a method "`notifyAlarm()`" is decided at **execution time**. (not at compilation time)

19

Outline

- Object Oriented Programming
- Data Encapsulation
- Inheritance
- Polymorphism
- ***Using Abstract and Interface***

20

Abstract

- A special kind of class that **cannot be instantiated.**
- It allows only other classes to **inherit from** it.
- It **enforces certain hierarchies** for all the subclasses

21

Interface

- An Interface is **NOT** a class.
- An Interface has **NO** implementation inside.
  - Definitions of methods without body.

22

Comparison-1

Feature	Interface	Abstract Class
Multiple inheritance	A class may inherit **several** interfaces	Only **one**
Default implementation	**Cannot** provide any code	**Can** provide complete, default code and/or just the details that have to be overridden.
Access Modifier	**Cannot have access modifiers**. (everything is assumed as public)	**Can** have it.

23

Comparison-2

Feature	Interface	Abstract Class
Adding functionality (Versioning)	For a new method, we have to **track down all the implementations** of the interface and define implementation for the new method	For a new method, we can provide **default implementation** and all the existing code might work properly.
Fields and Constants	**No fields** can be defined in interfaces	**Fields and constants** can be defined

24

Next Reading

- 3.2 from Rosen
- Chapter 10 from Prichard

25