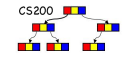# CS200: Advanced OO in Java

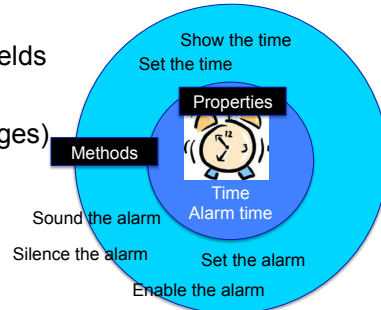## Object Oriented Programming

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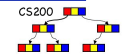- Interface?
- Package?

Basic Components

a software bundle of **related states** (**properties**, or **variables**) and **behavior** (**method**)

- State is stored in fields (variables in some programming languages)
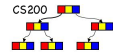- Method exposes object's behavior.

Show the time
Set the time

Properties

Methods

Time
Alarm time

Sound the alarm

Silence the alarm   Set the alarm

Enable the alarm

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.

1

Basic Components

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

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

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){…}
}
```
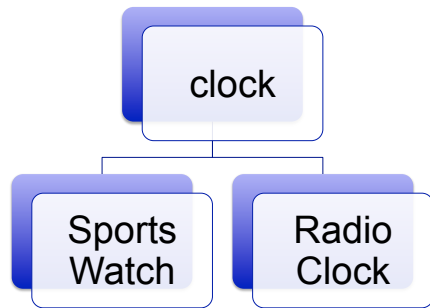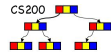
Inheritance

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

2

Example: Inheritance

```
      clock

Sports        Radio
Watch         Clock
```

Example: Inheritance – cont.

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

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

}
```
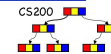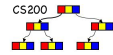
Overriding Methods

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

Java Access Modifiers

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

Polymorphism

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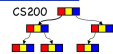
Polymorphic method

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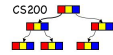
A: Clock
B. RadioClock

Dynamic Binding

- The version of a method "`notifyAlarm()`" is decided at **execution time**. (not at compilation time)

Abstract

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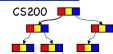- It **enforces certain hierarchies** for all the subclasses

Interface

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

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.

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