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

- **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

- **State**: stored in fields (variables in some programming languages)
- **Method**: exposes object’s behavior.
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

```java
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.
Example: Inheritance

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

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

Example: Inheritance – cont.

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
An Interface is **NOT** a class.

An Interface has **NO** implementation inside.
- Definitions of methods without body.

### Comparison-1

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

### Comparison-2

<table>
<thead>
<tr>
<th>Feature</th>
<th>Interface</th>
<th>Abstract Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adding functionality</td>
<td>For a new method, we have to track down all the implementations of the interface and define implementation for the new method</td>
<td>For a new method, we can provide default implementation and all the existing code might work properly.</td>
</tr>
<tr>
<td>Fields and Constants</td>
<td>No fields can be defined in interfaces</td>
<td>Fields and constants can be defined</td>
</tr>
</tbody>
</table>