Human Computer Interaction

UNIT-6
Lecture 4:
Cognitive Architecture

Mr. Nachiket Sainis

B K Birla Institute of Engineering & Technology, Pilani

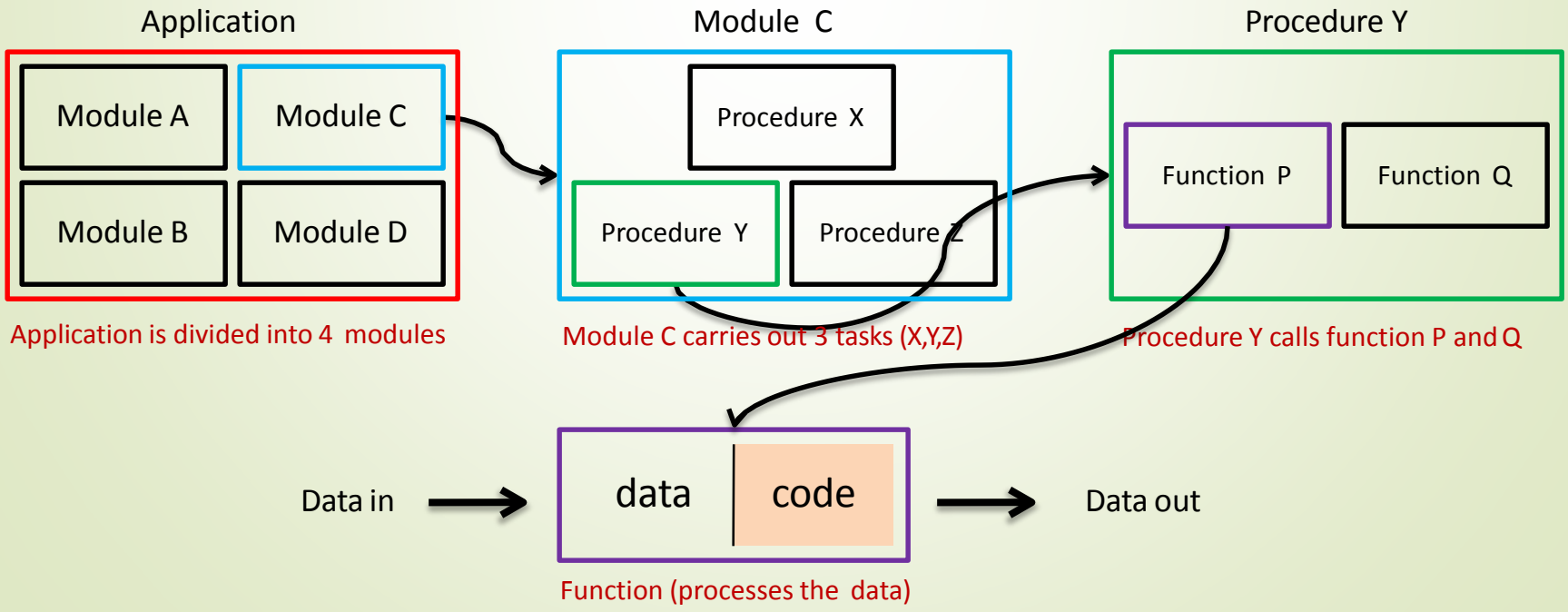
Object Oriented Programming

Overview

- Procedural Programming Paradigm - PPP
- Object-Oriented Programming Paradigm - OPP
- Objects
  - State and Behavior
  - Software Objects - Fields and Methods
  - Data Encapsulation
  - Public Interface
  - Object-based application
  - Benefits of object-based application development
- Classes
  - Definition
  - Blueprint of similar object
  - Instantiating objects from class - example
- Object-Oriented Principles
  - Encapsulation
  - Inheritance and Interfaces
  - Polymorphism
- Packages and API

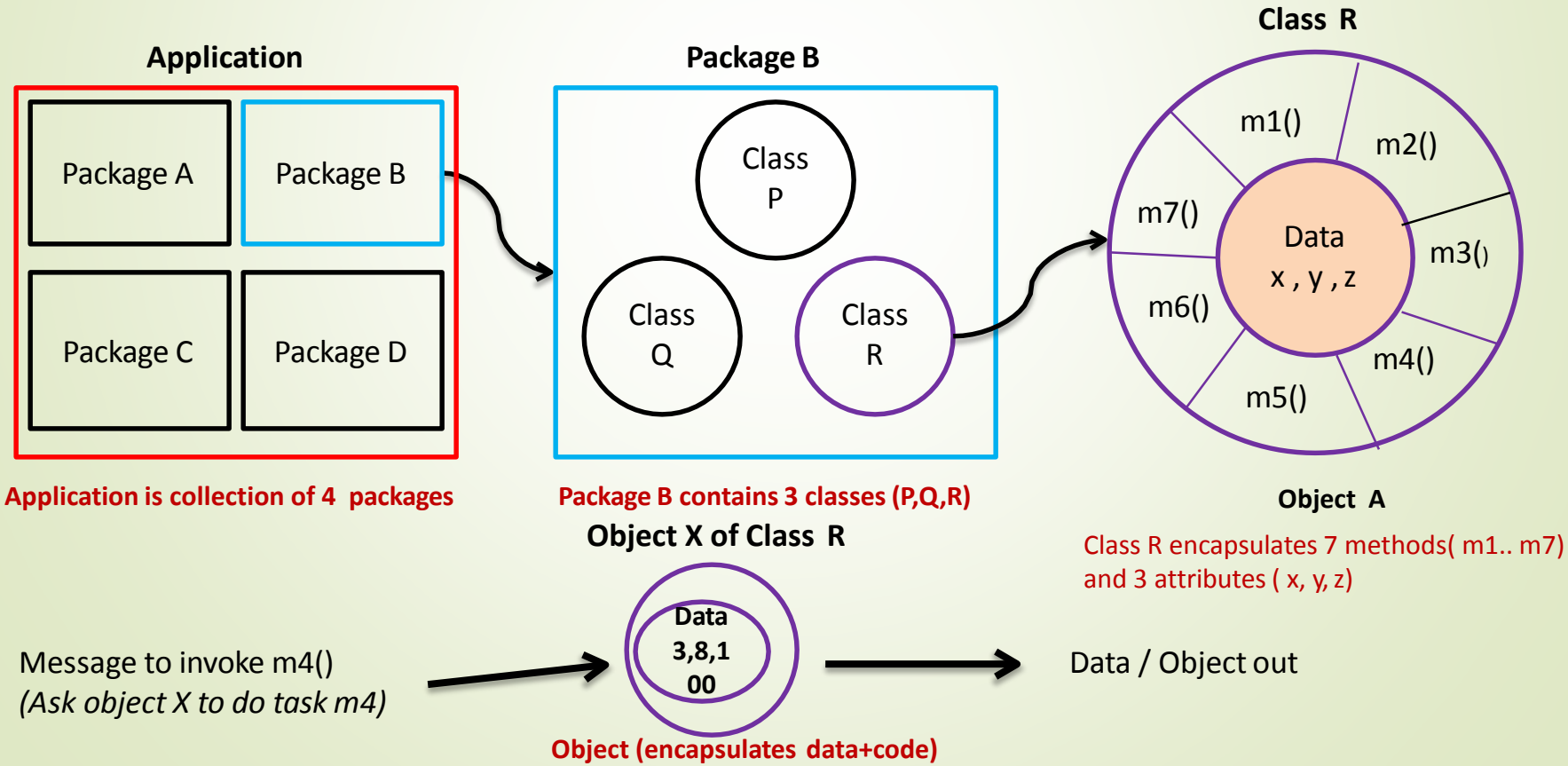
Procedural Programming Paradigm (PPP)

- In this paradigm (functional paradigm) the programming unit is a *function.*
- Program is divided into self-contained and maintainable parts called 'modules' (e.g. A,B,C,D).
- A module contains procedures - self-contained code that carries out a single task (e.g. X,Y,Z).
- A function (e.g. P, Q) is a self-contained code returning value to the calling procedure (e.g. Y).
- In PPP all computations are done by applying functions.
- Procedural paradigm separates data of program from instructions that manipulate the data.



Object Oriented Programming Paradigm (OOPP)

- Paradigm describes a system as it exists in real life based on interactions among real objects.
- Application is modeled as collection of related *objects* that interact and do your work (task).
- The programming unit is a *class* representing collection of similar real world objects.
- Programming starts with abstraction of useful real world objects and classes.
- Application is divided into multiple packages (e.g. A,B,C,D).
- A package is a collection of classes (e.g. P,Q,R) fulfilling group of tasks or functionality.
- A class is an encapsulated(data + code) definition of collection of similar real world objects.
- OOPP binds together data and the instructions that manipulate the data into a class.



Objects

Objects are key to understanding *object-oriented* technology. Look around right now and you'll find many examples of real-world objects

Objects

Real-world objects share two characteristics: They all have ***state*** **and** ***behavior***

DOG

State	Name Color Breed Hungry
Behavior	Barking Fetching Wagging Tail

BICYCLE

State	Current Gear Current Speed Current Pedal Cadence (rhythm)
Behavior	Changing gear Changing pedal cadence Applying brakes

Objects

Try and identify the State & behavior of the two objects .

TABLE LAMP

State	?
Behavior	?

DVD PLAYER

State	?
Behavior	?

Answer in the next slide

Objects

TABLE LAMP

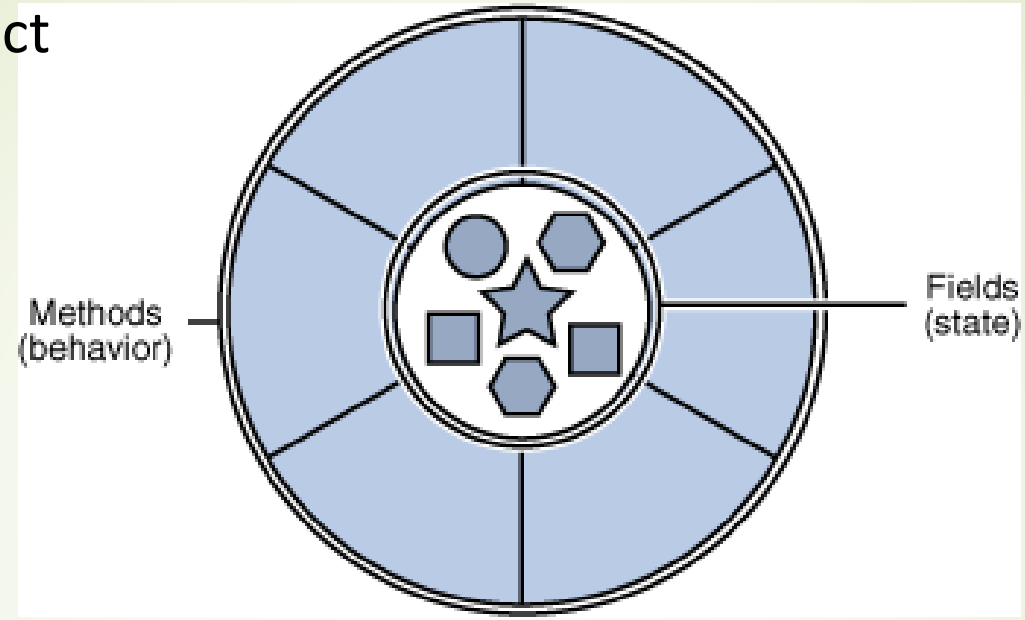
State	On/Off
Behavior	Turning On Turning Off

DVD PLAYER

State	On /Off Current Volume Current Station
Behavior	Turn on Turn off Increase volume Decrease volume, Seek Scan Tune

Software Objects - fields and methods

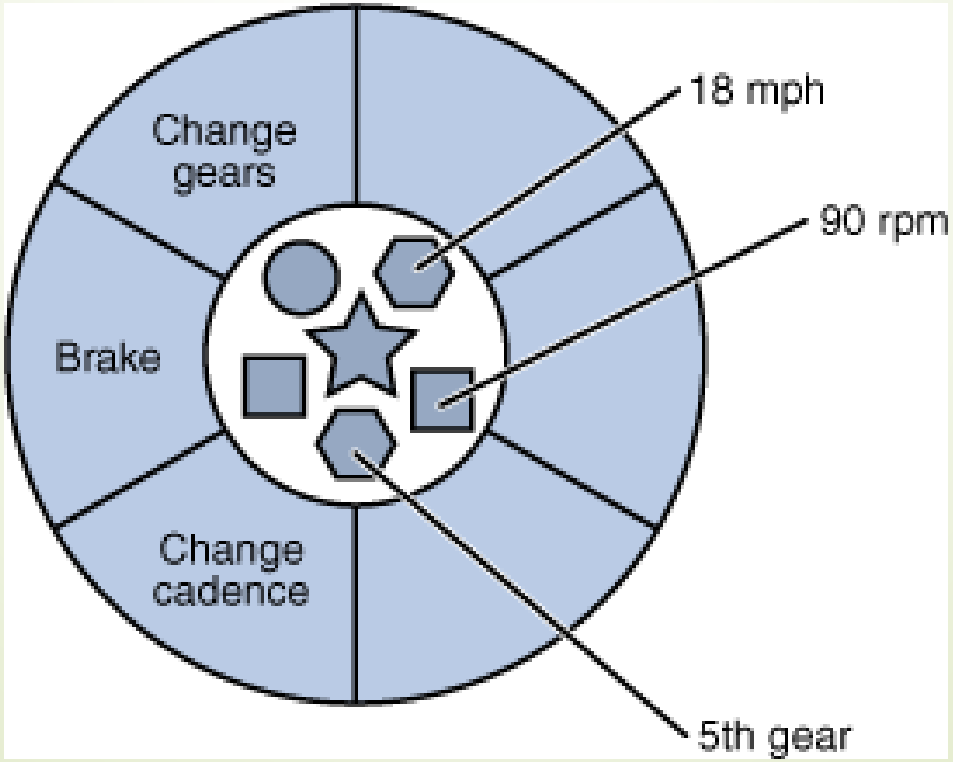
Software Object



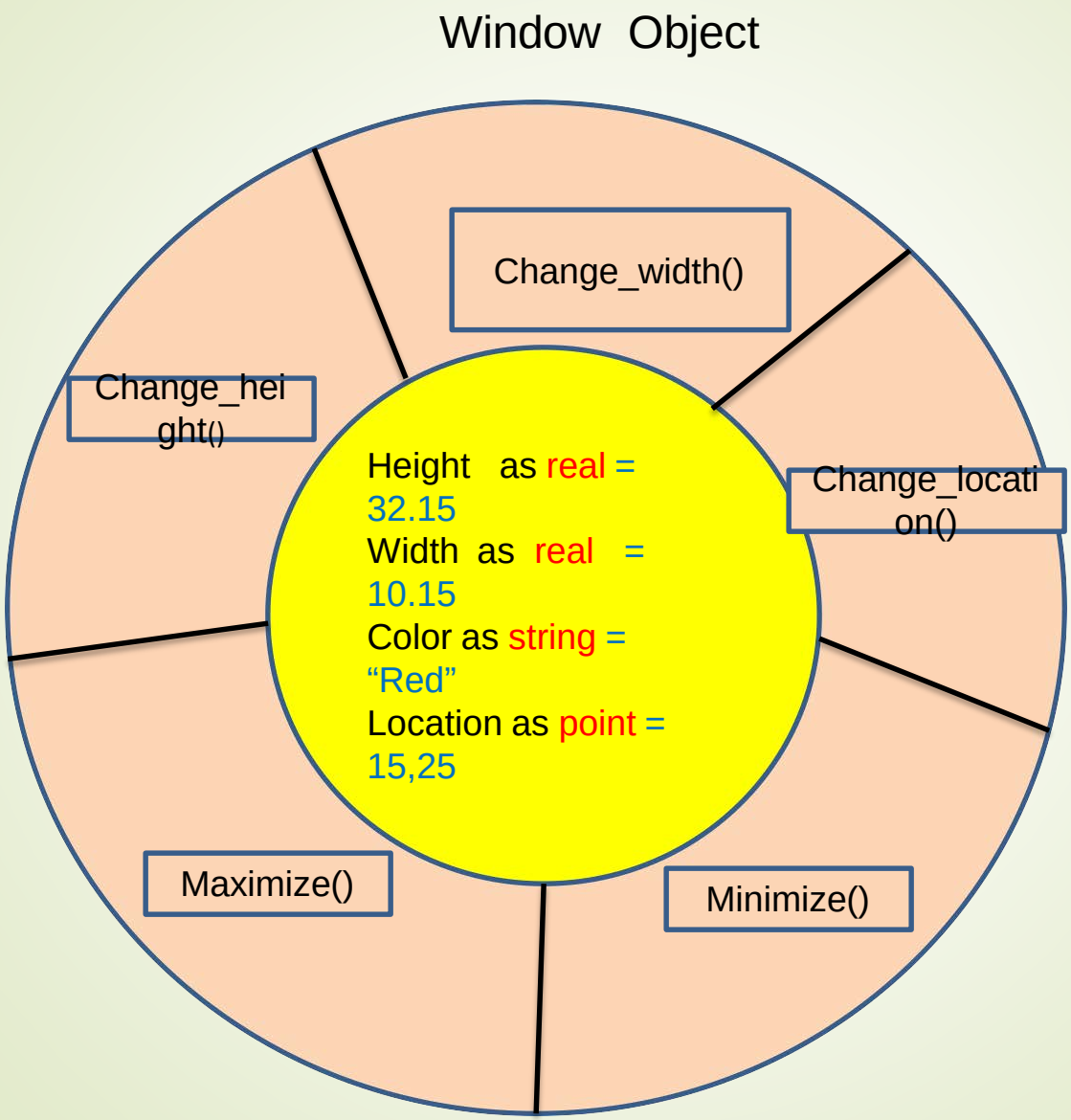
An object stores its state / information in ***fields*** (attributes) and exposes its behavior through ***methods*** (member functions)

Objects - Data Encapsulation

Objectsmethods can only change object's internal state. Hiding this internal state & requiring all interaction to be performed through an object's methods is known as ***data encapsulation***



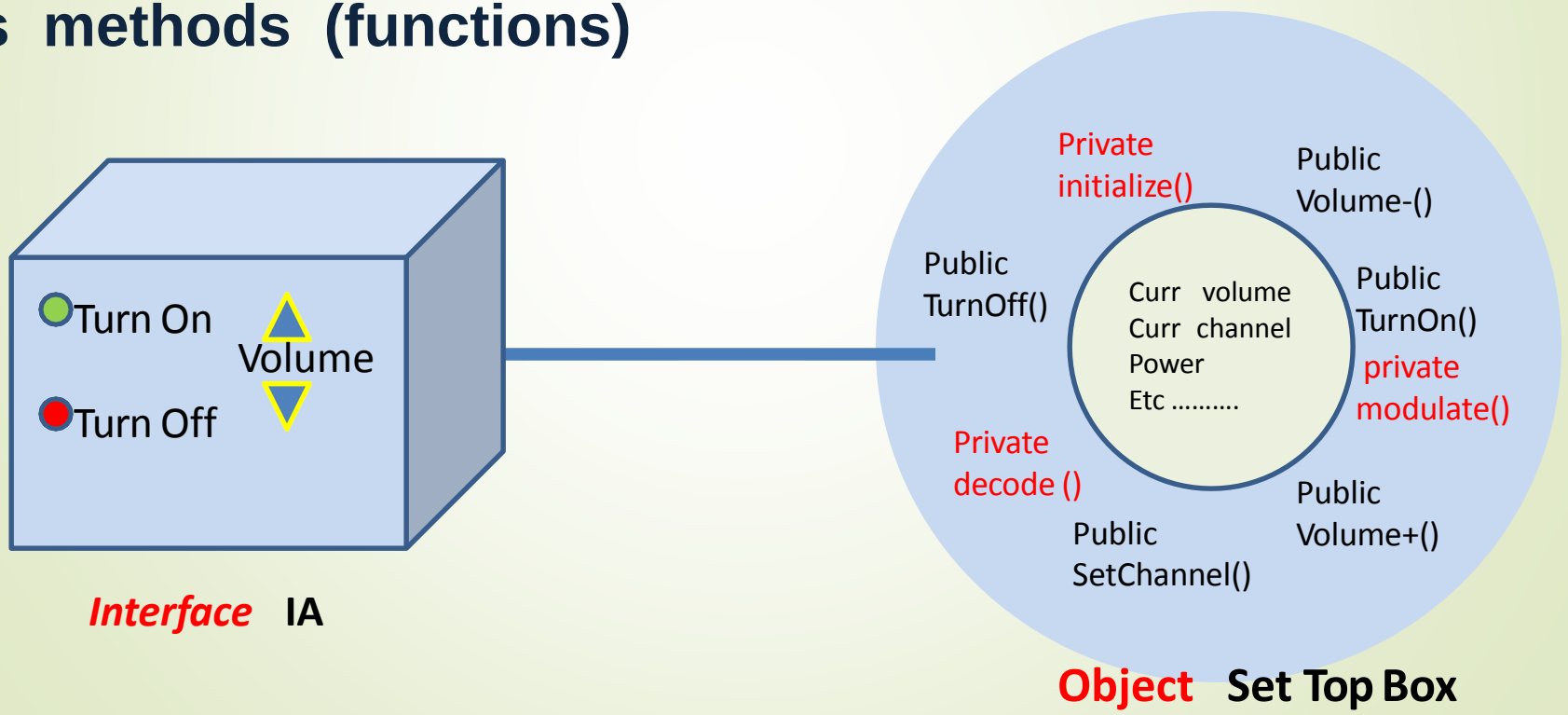
Objects – Data Encapsulation



Objects - Public Interface

Other objects can change the state of an object by using only those methods that are exposed to the outer world through a public interface. This help in data security.

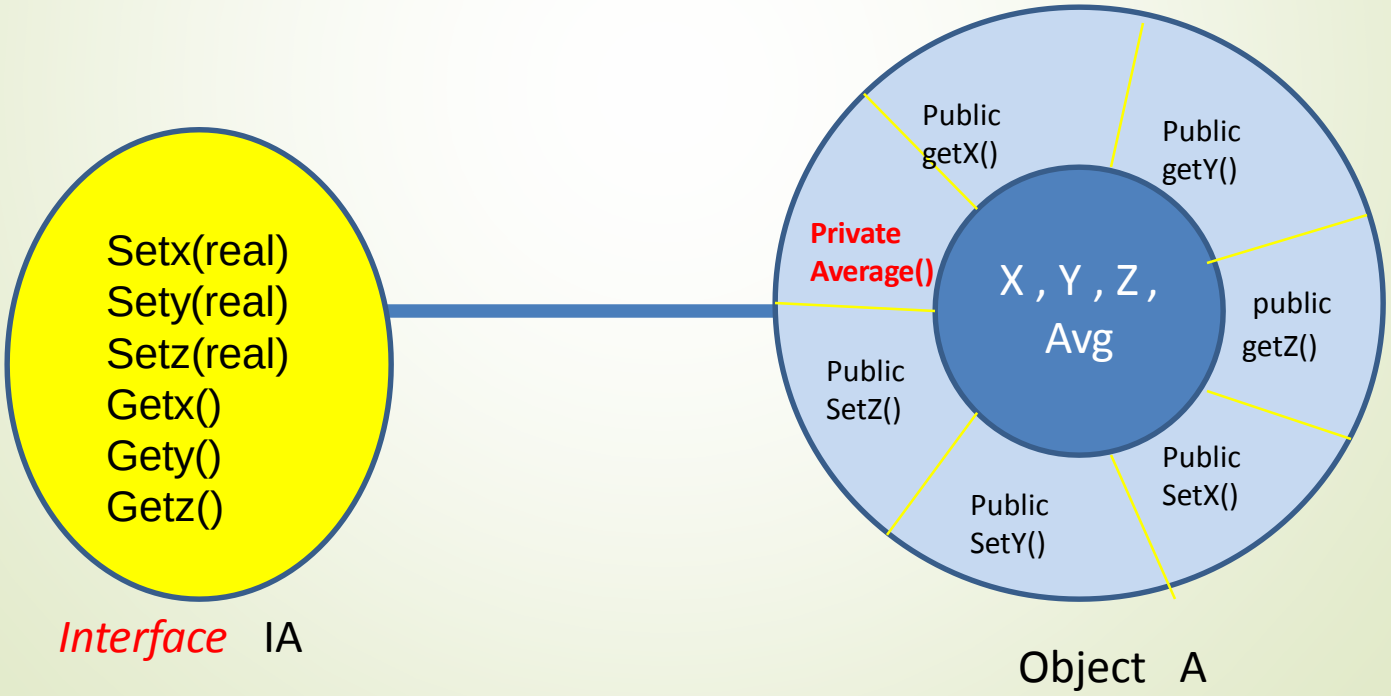
Object is an encapsulation of **data (attributes) as well as methods (functions)**



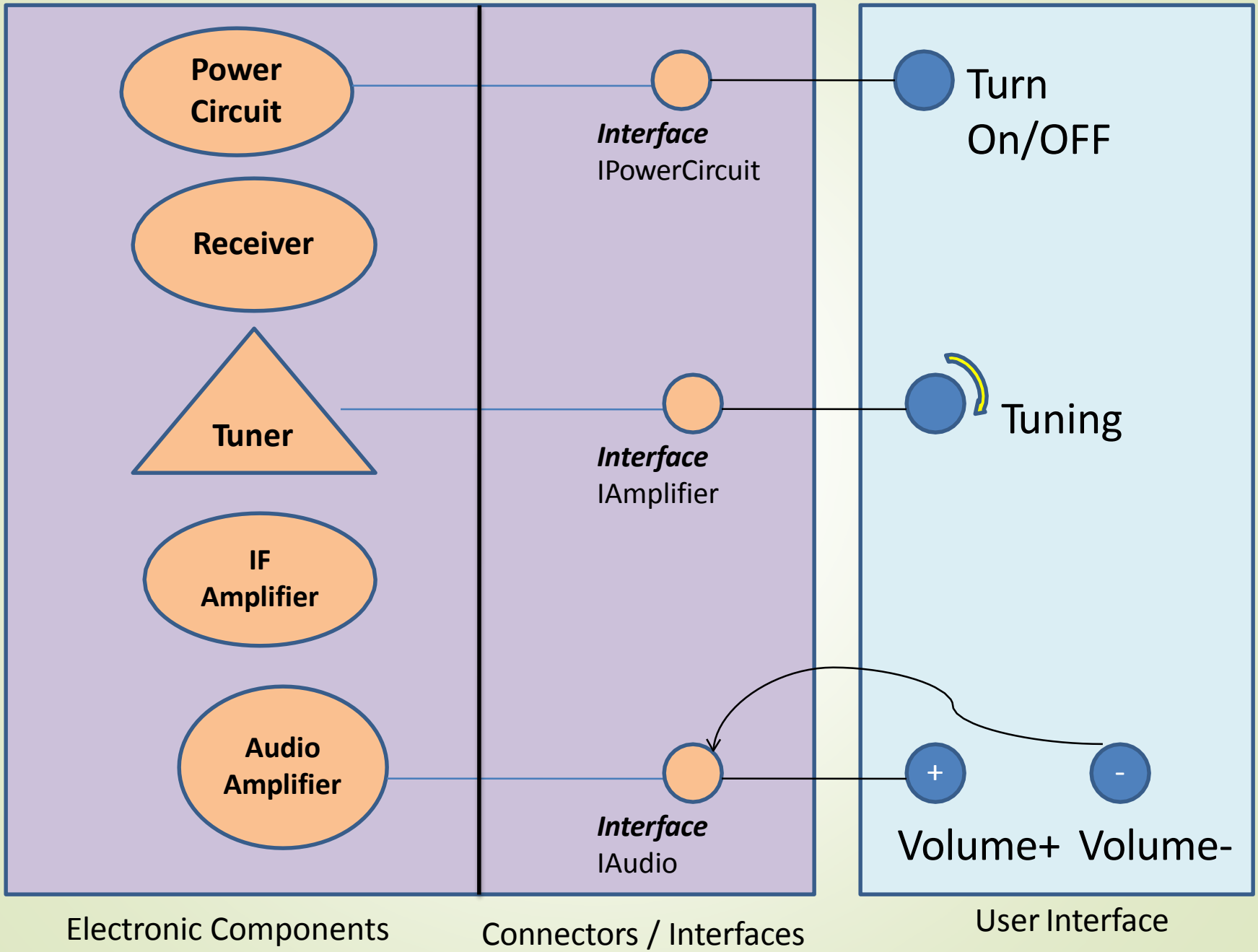
Objects - Public Interface

Other objects can change the state of an object by using only those methods that are exposed to the outer world through an public interface. This help in data security.

Object is an encapsulation of **data (attributes) as well as methods (functions)**



Object-based application



Benefits of object-based application development

- **Modularity:**

  Source code for an object can be maintained independently in a class. Once created, an object can be easily passed around inside the system.

  E.g. ClassMaths, Class string , ClassWindow etc

- **Information-hiding:**

  By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world. E.g. *Interface* IMaths , *Interface* IString *Interface* IWindow

- **Code re-use:**

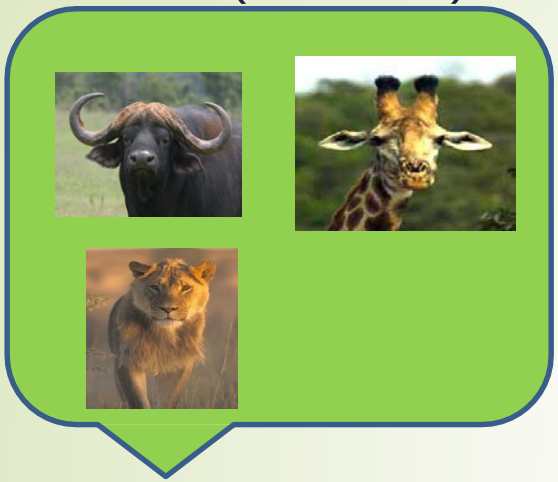
  If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.

  e.g. Standard Classes available packages which can reused using their interfaces.

- **Pluggability and debugging ease:** If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.

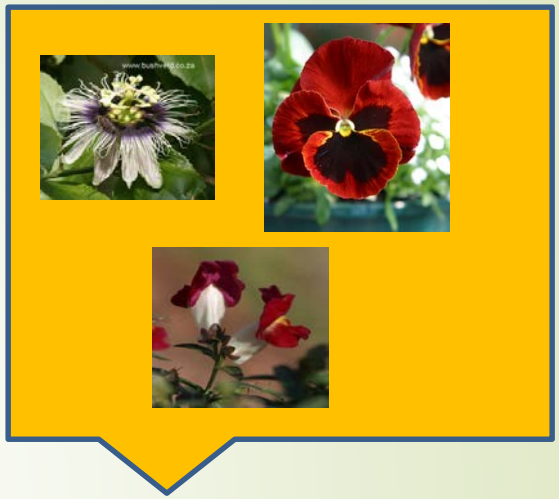
Class

Collection of objects that share common attributes and behavior (methods)

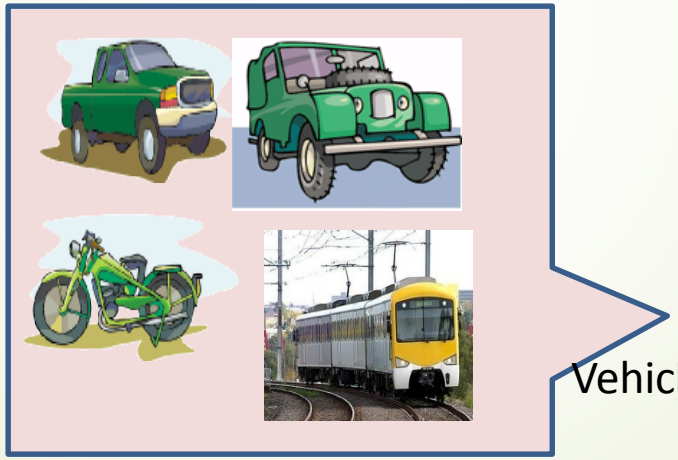


Animals

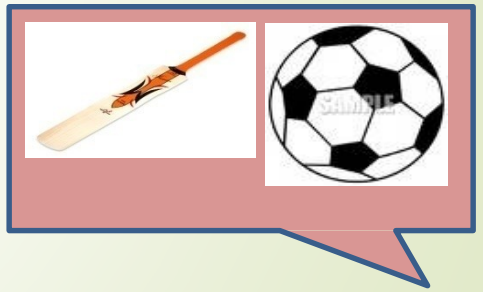
Airplanes



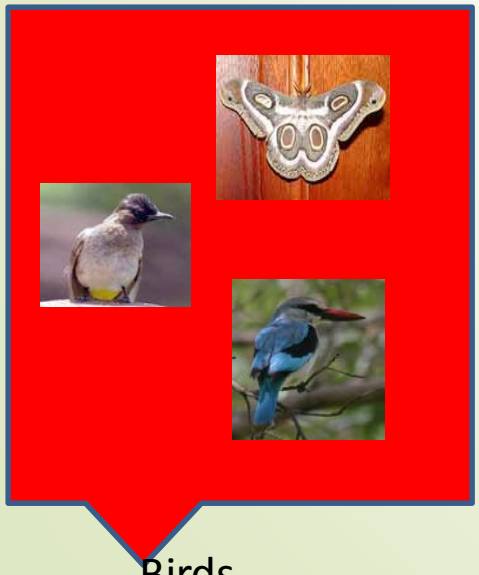
Flowers



Vehicles



Animals



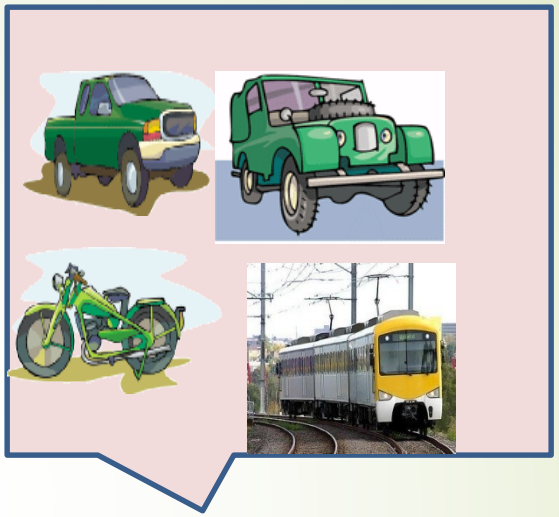
Birds

Humans

Stationary

Class - Structure

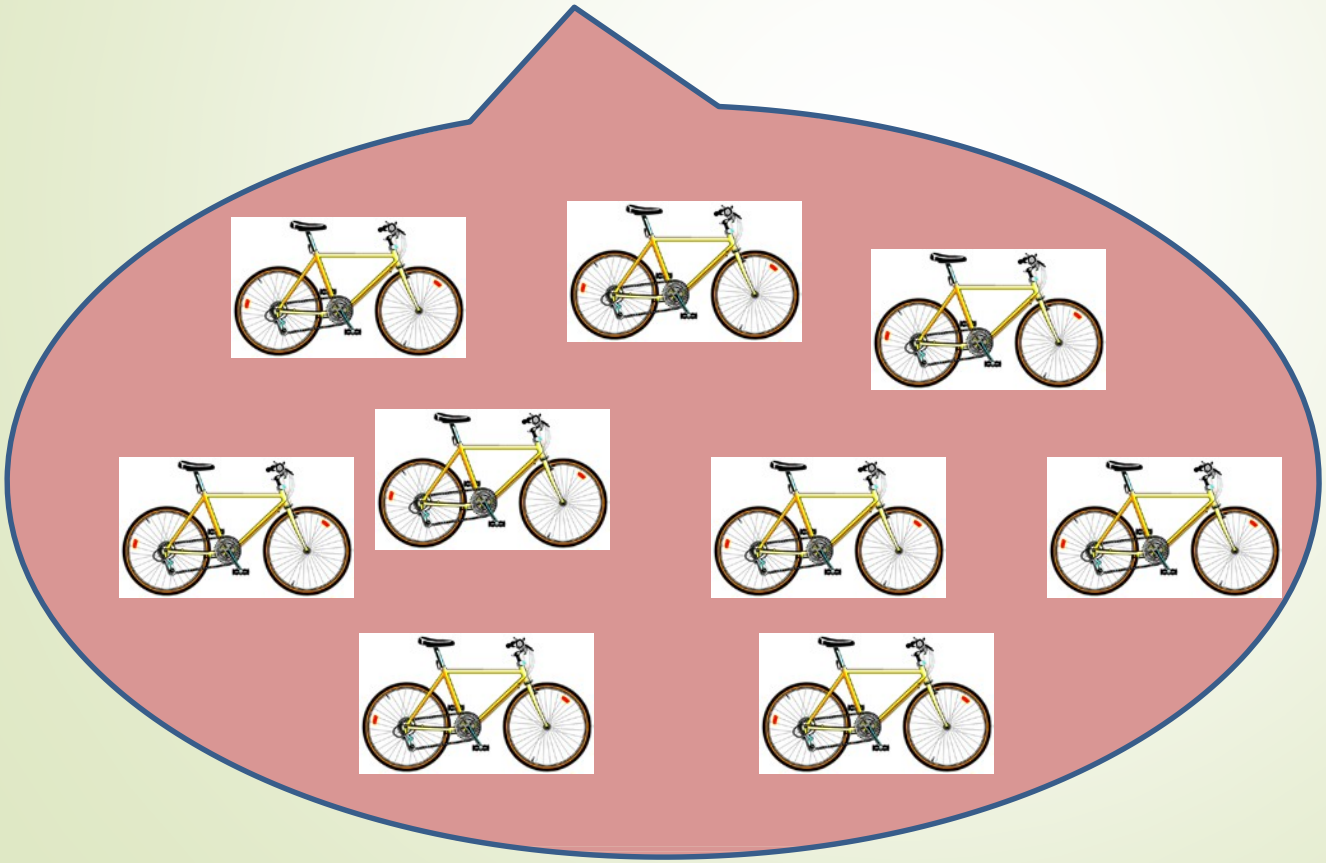
Class Animal
Attributes
Behavior



Class Vehicle
Attributes
Behavior

Class - Blueprint of similar object

Factory of cycles of same make and model with same features and same components . Built from same blueprint. This blueprint is called a Class.



In object-oriented terms, we say that your cycle is an ***instance* of the *class* of objects** known as **cycles**

A *class* is the blueprint from which individual objects are created.

The attributes and operations defined by a class are for its objects, not for itself.

A class is a logical construct, an object has physical reality.

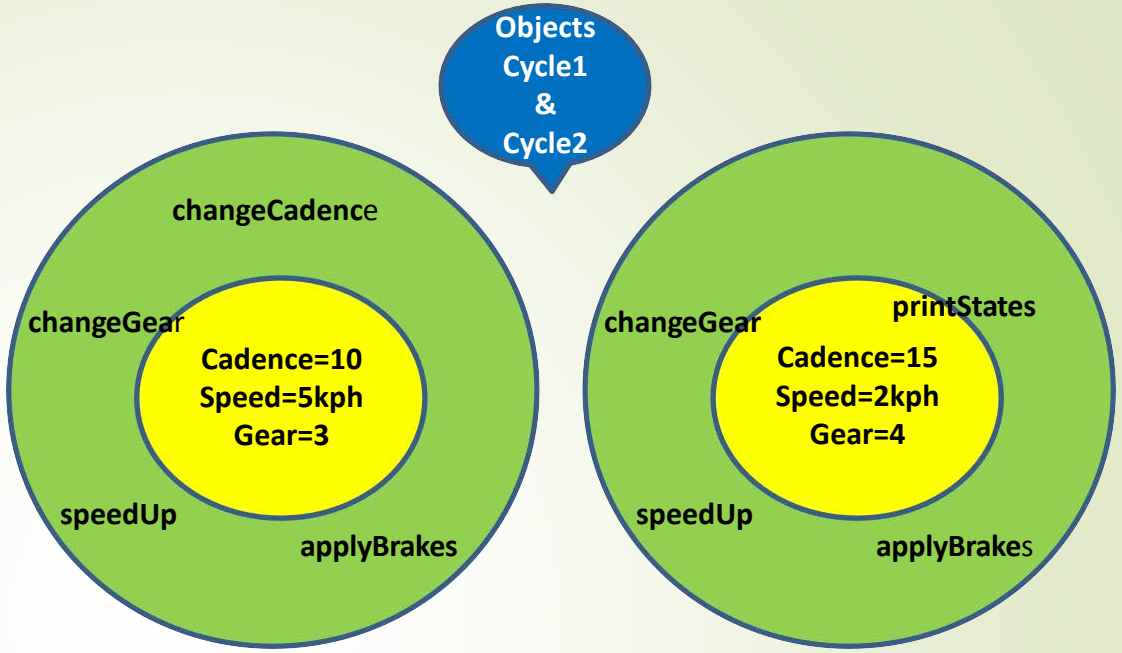
Class Definition – Example

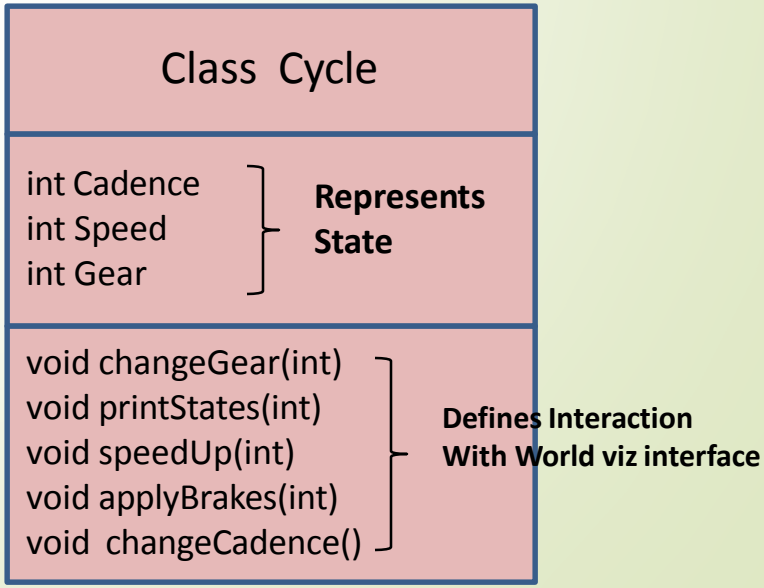
Classes are passive and which do not communicate but are used to create (instantiate) objects which interact.

The responsibility of creating and using new Cycle objects belongs to some other class in your application.

```
class Cycle
    {           int cadence = 0;
                int speed = 0;
                int gear = 1;

          void changeCadence(int newValue)
                { cadence = newValue; }

          void changeGear(int newValue)
                { gear = newValue; }

          void speedUp(int increment)
                { speed = speed + increment; }

          void applyBrakes(int decrement)
                { speed = speed - decrement; }

          void printStates()
            { System.out.println("cadence:"+cadence+"
              speed:"+speed+" gear:"+gear); }
    }
```

cycle1 and cycle2 are instances of **Class Cycle**





Instantiating objects from class

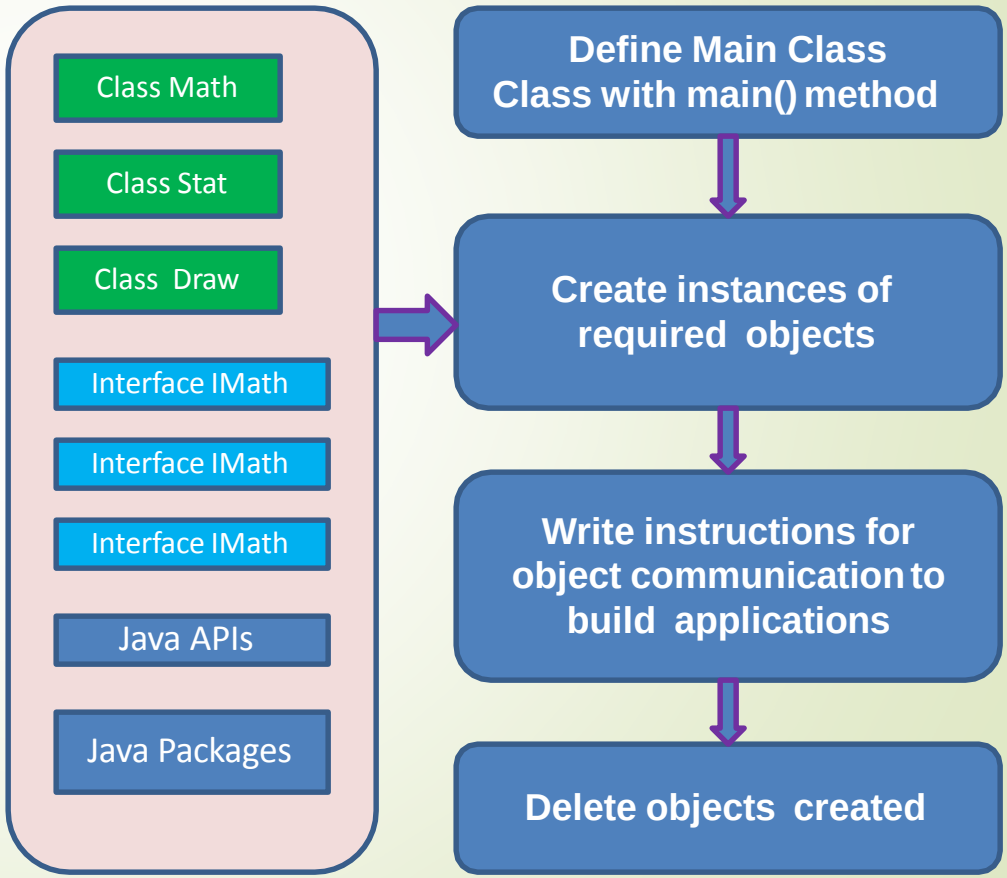
Cycle Demo class that creates two separate cycle objects and invokes their methods:

```
Class CycleDemo
{
    public static void main(String[] args)
        { // Create two different Bicycle
objects
                Bicycle bike1 = new Bicycle();
                Bicycle bike2 = new Bicycle();

          // Invoke methods on those objects
                bike1.changeCadence(50);
                bike1.speedUp(10);
                bike1.changeGear(2);
                bike1.printStates();

                bike2.changeCadence(50);
                bike2.speedUp(10);
                bike2.changeGear(2);
                bike2.changeCadence(40);
                bike2.speedUp(10);
                bike2.changeGear(3);
                bike2.printStates();
        }
}
```

This program is stored in file CycleDemo.java

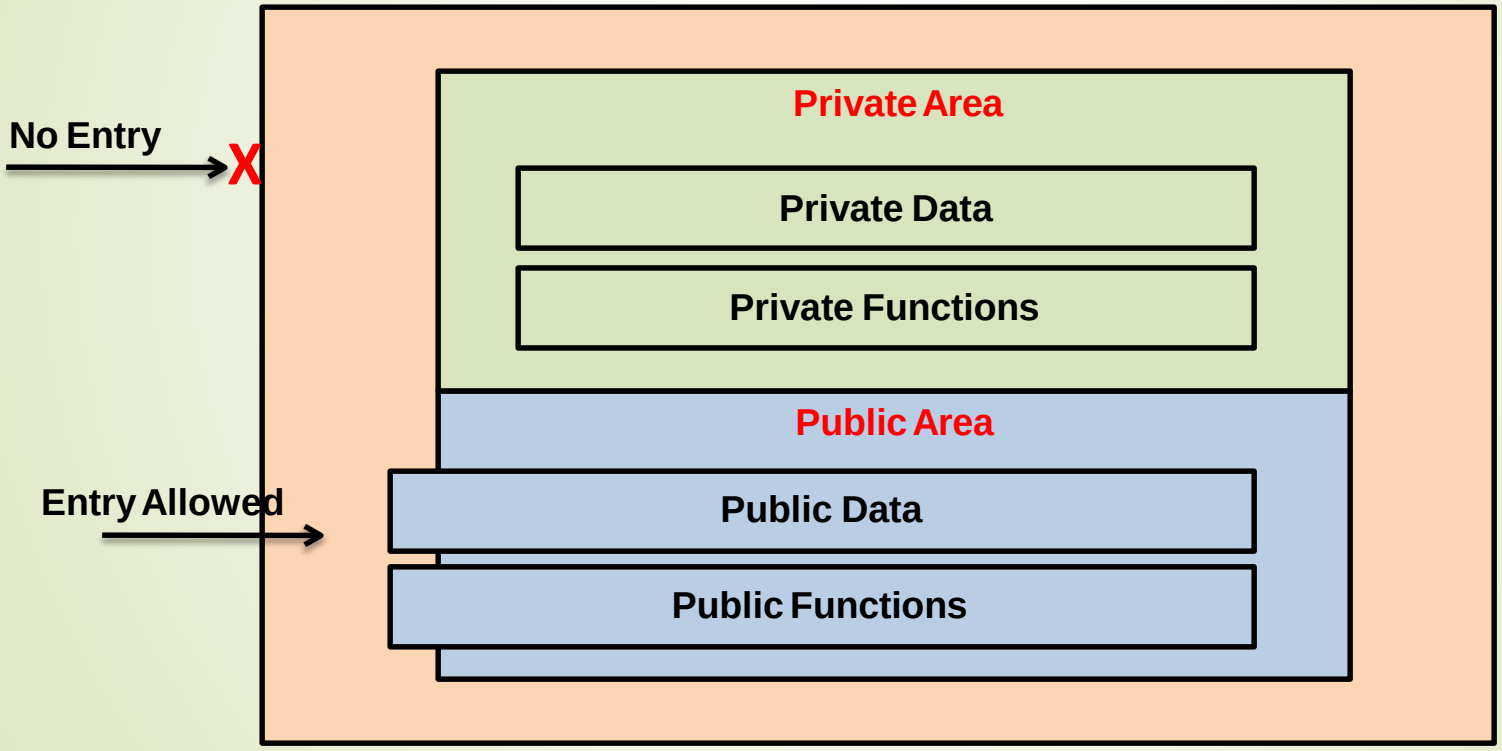
Main Class - Is a class that contains one method, called main. The main method contains a series of instructions to create objects and to tell those objects to do things.

Setting an Application's Entry Point



Object-Oriented Principle – Encapsulation

- Encapsulation is the mechanism that binds together the code and the data it manipulates, and keeps both safe from outside interference and misuse

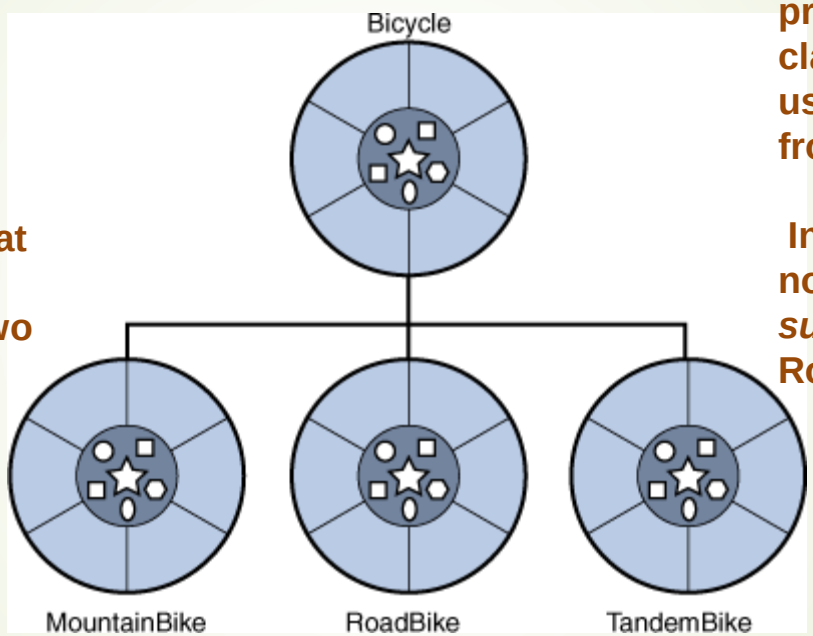


Object-Oriented Principle – Inheritance

Inheritance is the process by which one object acquires the properties of another object. By use of inheritance, an object need only define all of its characteristics that make it unique within its class, it can inherit its general attributes from its parent. This will be clear from the following example

Mountain bikes, road bikes, and tandem bikes all share the characteristics of bicycles (current speed, current pedal cadence, current gear).

Yet each has some features that makes them different: tandem bicycles have two seats and two sets of handlebars; road bikes have drop handlebars; some mountain bikes have an additional chain ring, giving them a lower gear ratio.



Object-oriented programming allows classes to *inherit* commonly used state and behavior from other classes.

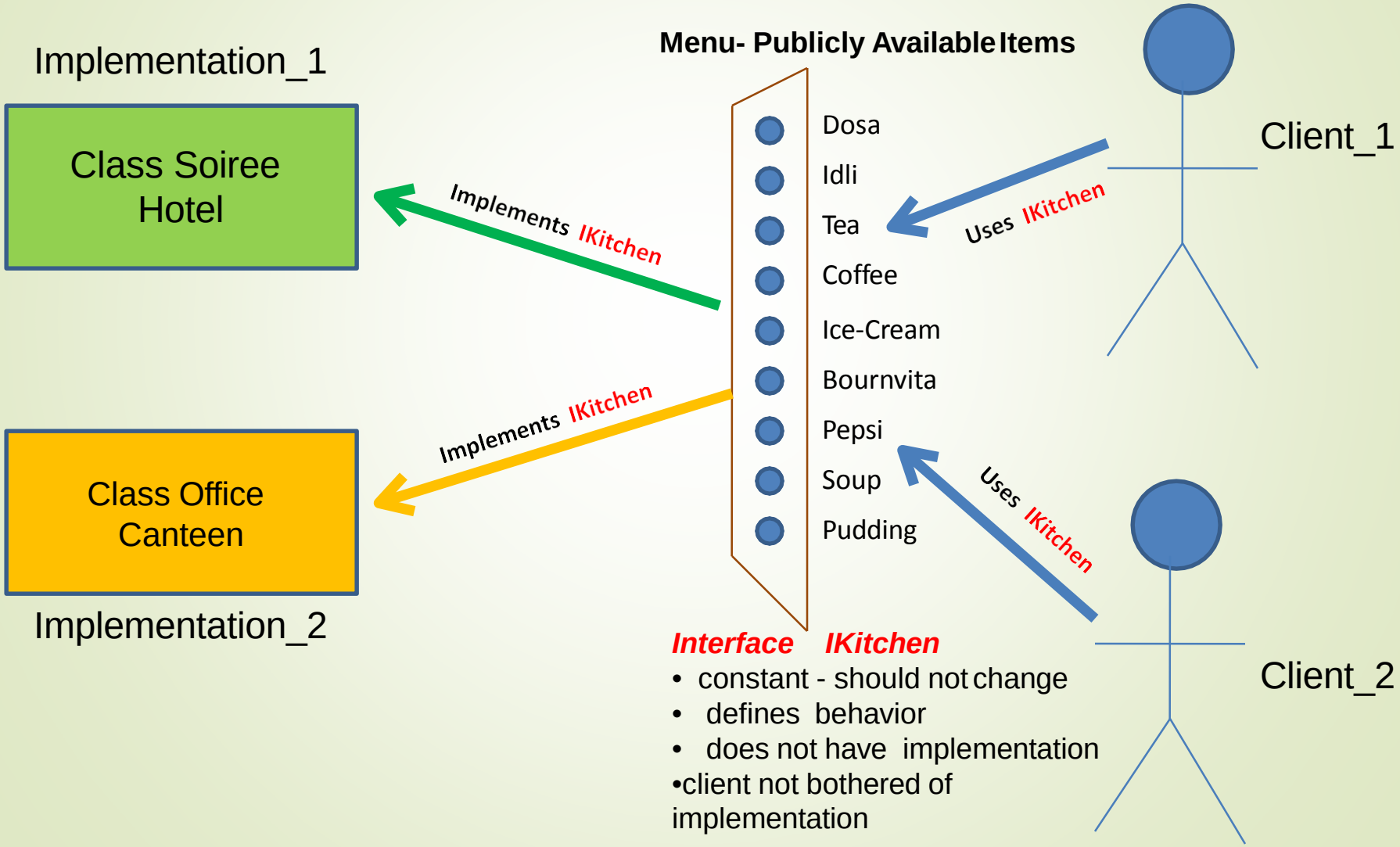
In this example, Bicycle now becomes the *superclass* of MountainBike, RoadBike, and TandemBike.

Creating subclass - at beginning of *class* use the extends keyword, followed by name of the class to inherit from:

```
class MountainBike extends Bicycle
  {
      // new fields and methods defining a mountain bike would go here
  }
```

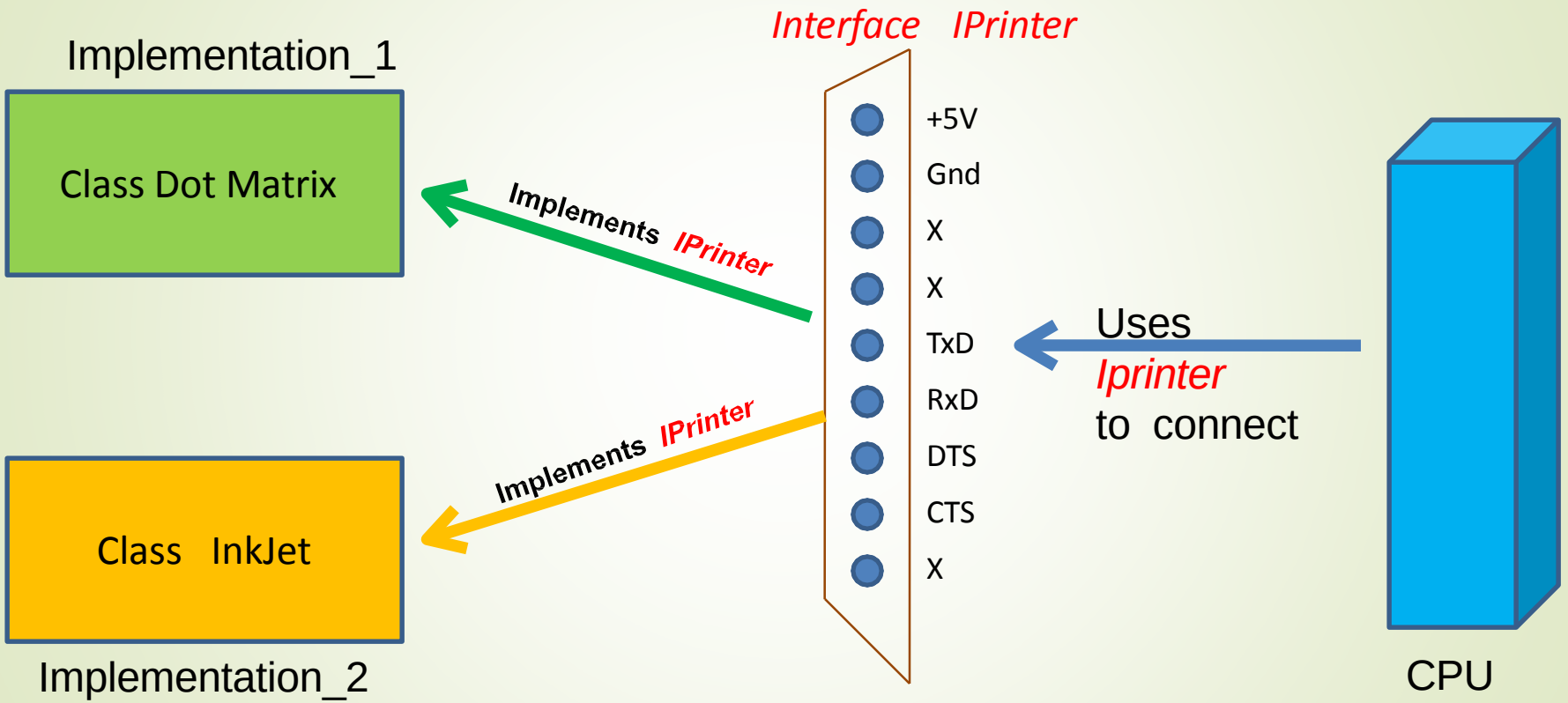
Interfaces

- Public methods form the object's ***interface*** with the outside world
- Interface is a group of related methods with empty bodies (pure virtual functions)



Interfaces

- Public methods form the object's ***interface*** with the outside world
- Interface is a group of related methods with empty bodies (pure virtual functions)

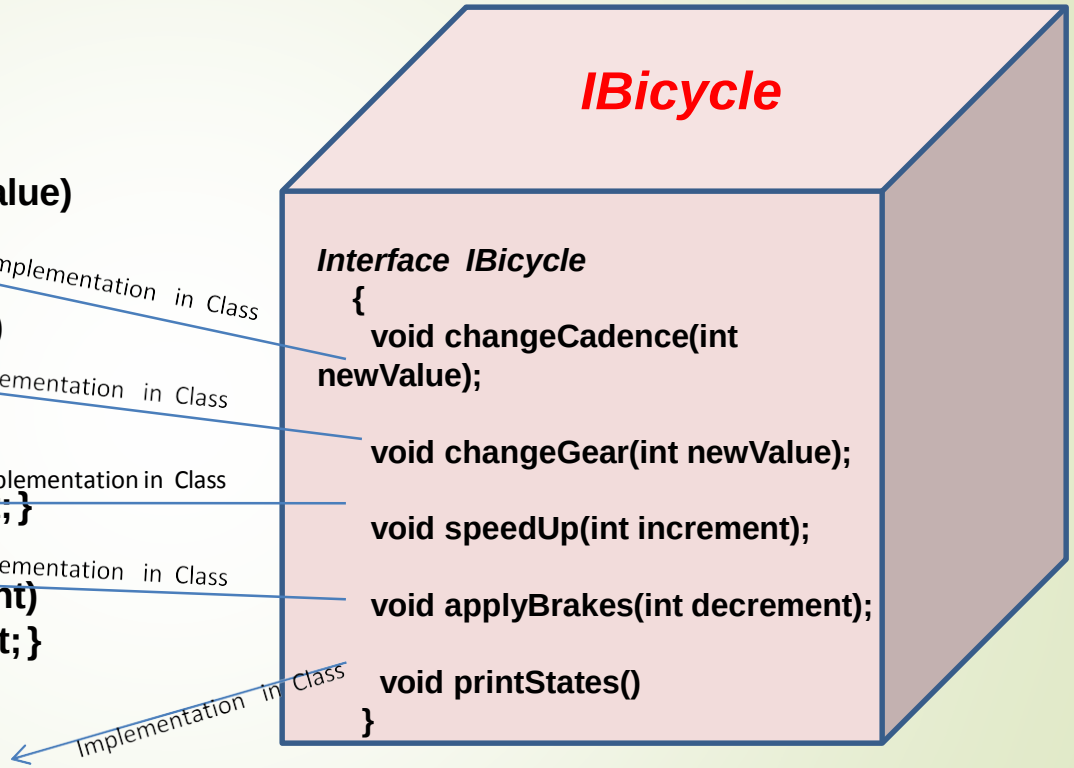


Class as implementation of interface

- A bicycle's behavior, if specified as an interface, might appear as follows

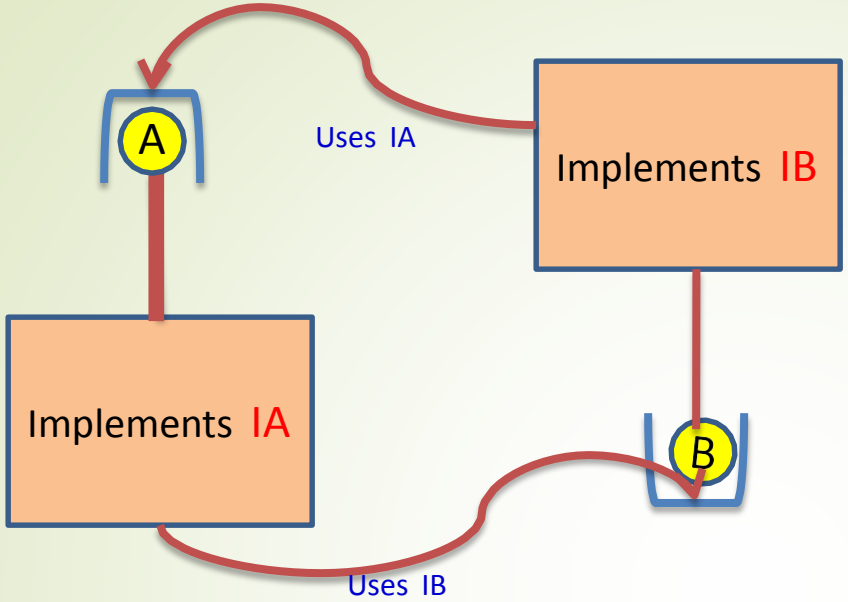
e.g. Class Soiree **implements** *IKitchen*

```
Class Bicycle implements  IBicycle
{
        int cadence = 0;
        int speed = 0;
        int gear = 1;

        void changeCadence(int newValue)
            { cadence = newValue; }

        void changeGear(int newValue)
           { gear = newValue; }

        void speedUp(int increment)
           { speed = speed + increment; }

         void applyBrakes(int decrement)
            { speed = speed - decrement; }

        void printStates()
          {
System.out.println("cadence:"+cadence+"
            speed:"+speed+" gear:"+gear); }

}
```



To implement interface, the name of your class would change (to a particular brand of bicycle, for example, such as ACMEBicycle), and you'd use the implements keyword in the class declaration

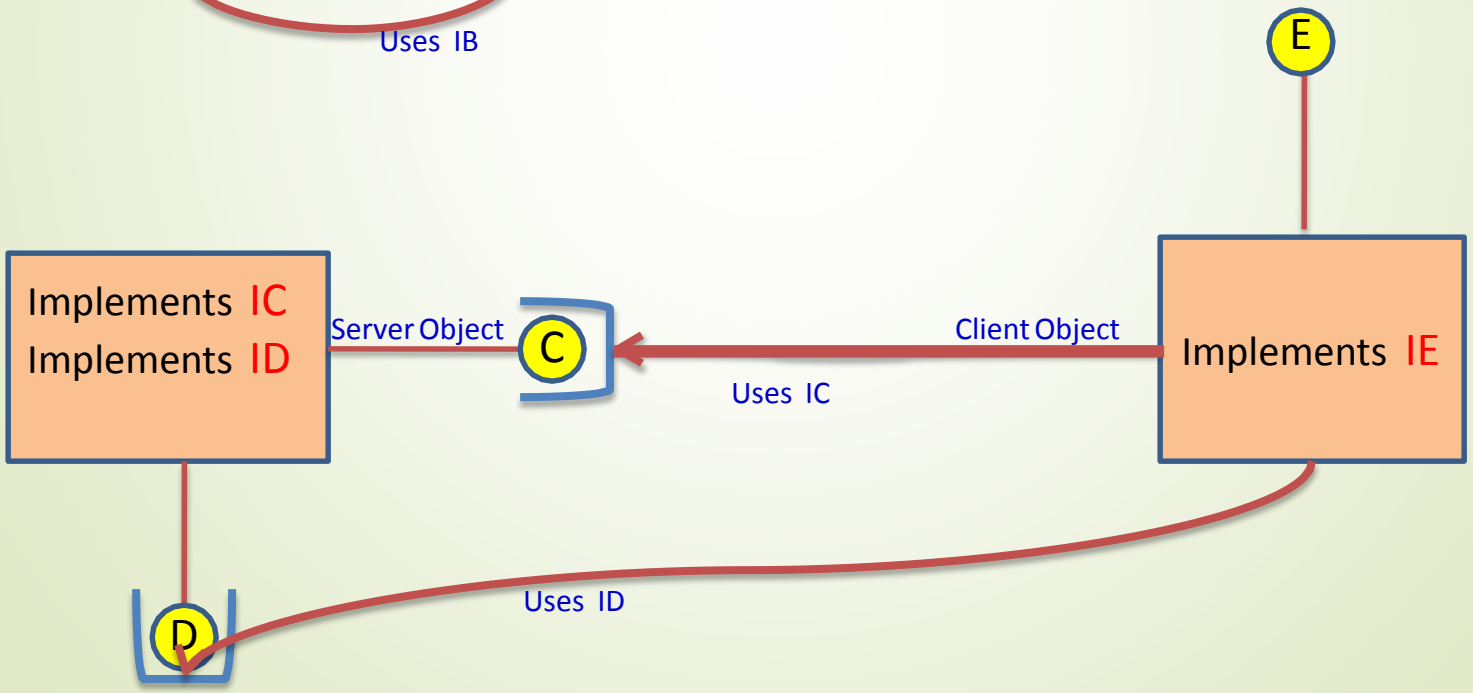
Interfaces define an application



•An interface makes a class formal about the behavior it promises to provide.

- Interfaces form a contract between the class and the outside world and this contract is enforced at build time by the compiler.

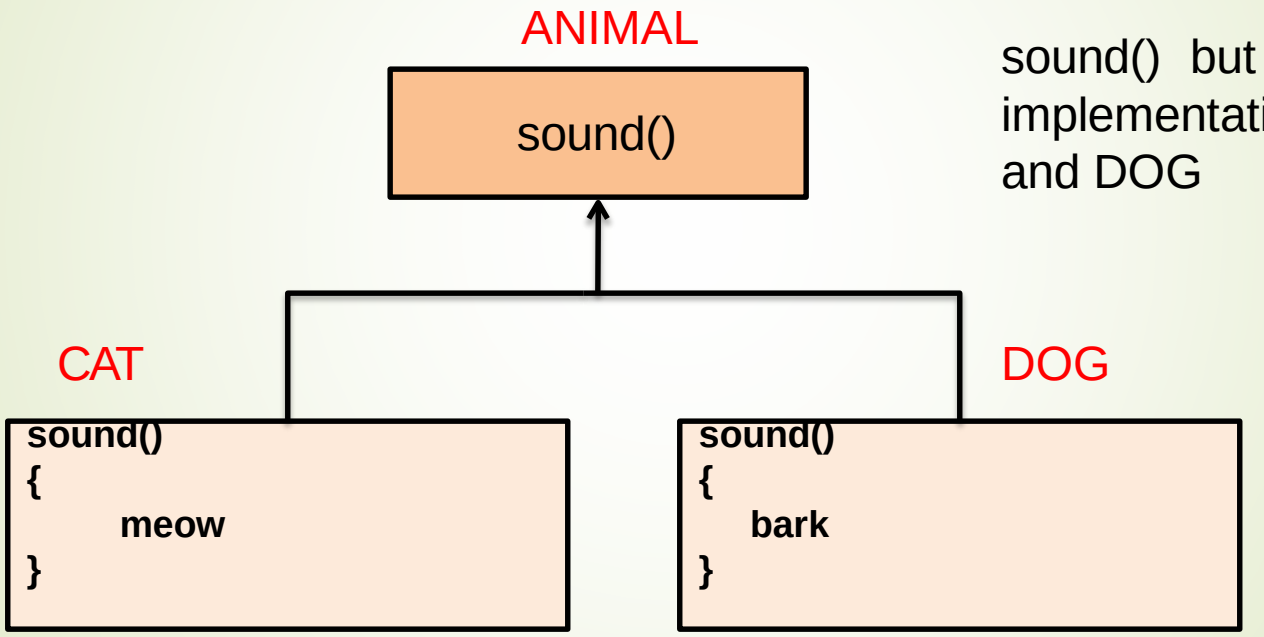
- If your class claims to implement an interface, all methods defined by that interface must appear in its source code before the class will successfully compile.



Object-Oriented Principle – Polymorphism

Polymorphism (from Greek, meaning 'many forms') is a feature that allows same interface (method name) to be used for a multiple class of actions depending on context.

Examples shows same name of method i.e.



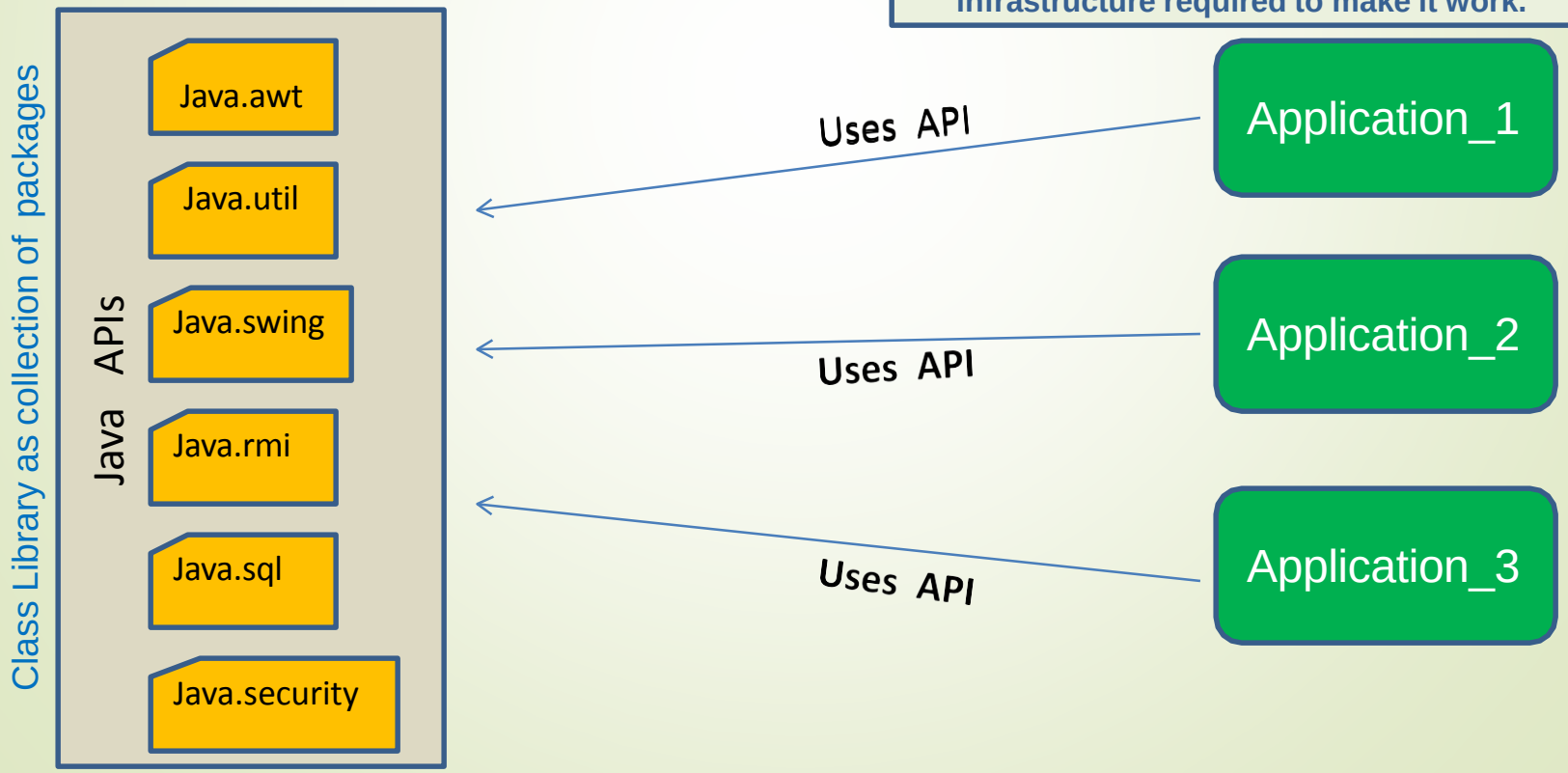
sound() but different implementations in CAT and DOG

Packages and API

- A package is a namespace that organizes a set of related classes and interfaces like a folder
- Software written in the Java programming language can be composed of hundreds or *thousands* of reusable classes, it makes sense to keep things organized by placing related classes and interfaces into packages.
- Java Platform provides Class library which has huge number of classes organized in packages which is also called API.

There are literally thousands of classes to choose from. This allows you, the programmer, to focus on the design of your particular application, rather than the infrastructure required to make it work.

Application Programming Interface (API)
(Standard nuts and bolts)



References

Books

- **The Object-Oriented Thought Process** by Matt Weisfeld, Publisher: Addison-Wesley Professional
- **Head First Object-Oriented Analysis & Design** by Gary Pollice, David West, Brett D McLaughlin , Publisher: Shroff O Reilly
- Experiencing Object Oriented Concepts: For Beginners by John E. Mathew

Ueful Links

http://scg.unibe.ch/archive/osg/Nier89aSurveyOfOOConcepts.pdf

http://www.codeproject.com/Articles/15874/Understanding-Object-Oriented-Concepts

http://www.slideshare.net/bgjeecourse/objectoriented-concepts-5576132

http://www.youtube.com/watch?v=3bMsY5a7cBo