CSC584 Enterprise Programming

Chapter 1 – Review of objet oriented programming

MARSHIMA MOHD ROSLI

COMPUTER SCIENCE

Chapter 1 Outline

Review of Object Oriented Programming Concepts

a) Object Oriented Programming Concepts
- Objects, classes, packages

b) Inheritance & Polymorphism concepts
- Inheriting instances fields and methods
- Method overriding
- Access levels – public, protected, private
- Abstract super classes and methods
- Interface

Forget programming for a while.

Think about the World and the things that are in it.

- What things are objects? What things are not objects?

List four objects and list four non-objects in the classroom.

Answer..

	Objects	Non-objects
1	A pen	The upper 37% of the pen
2	A computer keyboard	The air above the keyboard
3	A shoe	The color of the shoe
4	A desk	All desks in the world

An object is made of tangible material (the pen is made of plastic, metal, ink).

An object holds together as a single whole (the whole pen, not a fog).

An object has properties (the color of the pen, where it is, how thick it writes...).

An object can do things and can have things done to it.

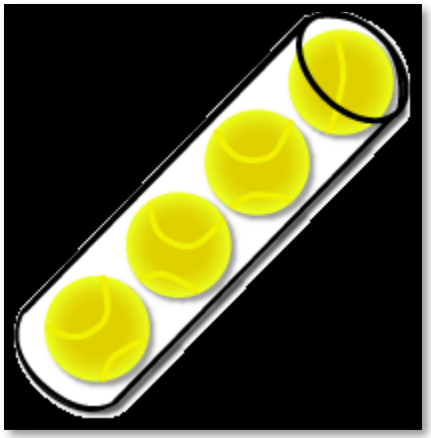
Characteristics of Objects

The last three items on the list seem clear enough. In fact, they have names:

- An object has **identity** (it acts as a single whole).
- An object has **state** (it has various properties, which might change).
- An object has **behavior** (it can do things and can have things done to it).

More about objects..

Consider a tube of four yellow tennis balls.



- Is the tube of tennis balls an object?
- Is each tennis ball an object?
- Could the top two balls be considered a single object?
- Is the color of the balls an object?

More about objects..

Is the tube of tennis balls an object?

- Yes. It has identity (my tube of balls is different than yours), it has state (opened, unopened, brand name, location), and behavior (although not much).

Is each tennis ball an object?

- Yes. It is OK for objects to be part of other objects. Although each ball has nearly the same state and behavior as the others, each has its own identity.

Could the top two balls be considered a single object?

- Not ordinarily. Each has its own identity independent of the other. If they were joined together with a stick you might consider them as one object.

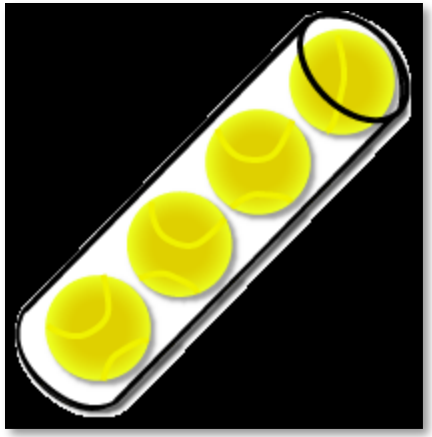
Is the color of the balls an object?

- No. It is a property of each ball.

Software Objects as Memory

Objects (real world and software) have *identity*, *state*, and *behavior*.

- Software objects have **identity**. Each is a distinct chunk of memory. (Just like a yellow tennis ball, each software object is a distinct individual even though it may look nearly the same as other objects.)



Software Objects as Memory

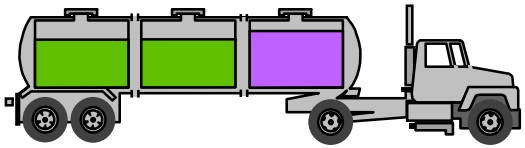
Objects (real world and software) have *identity*, *state*, and *behavior*.

- Software objects have **state.** Some of the memory that makes up a software object is used for variables which contain values. These values are the state of the object.
- Software objects have **behavior.** Some of the memory that makes up a software object contains programs (called *methods*) that enable the object to "do things". The object does something when one of its method runs.

What Is an Object?

Informally, an object represents an entity, either physical, conceptual, or software.

- Physical entity

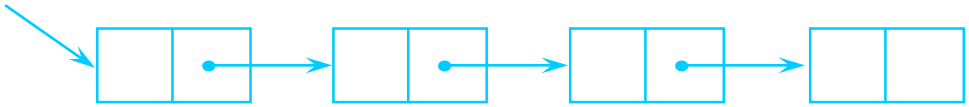


Truck

- Conceptual entity

Chemical Process

- Software entity

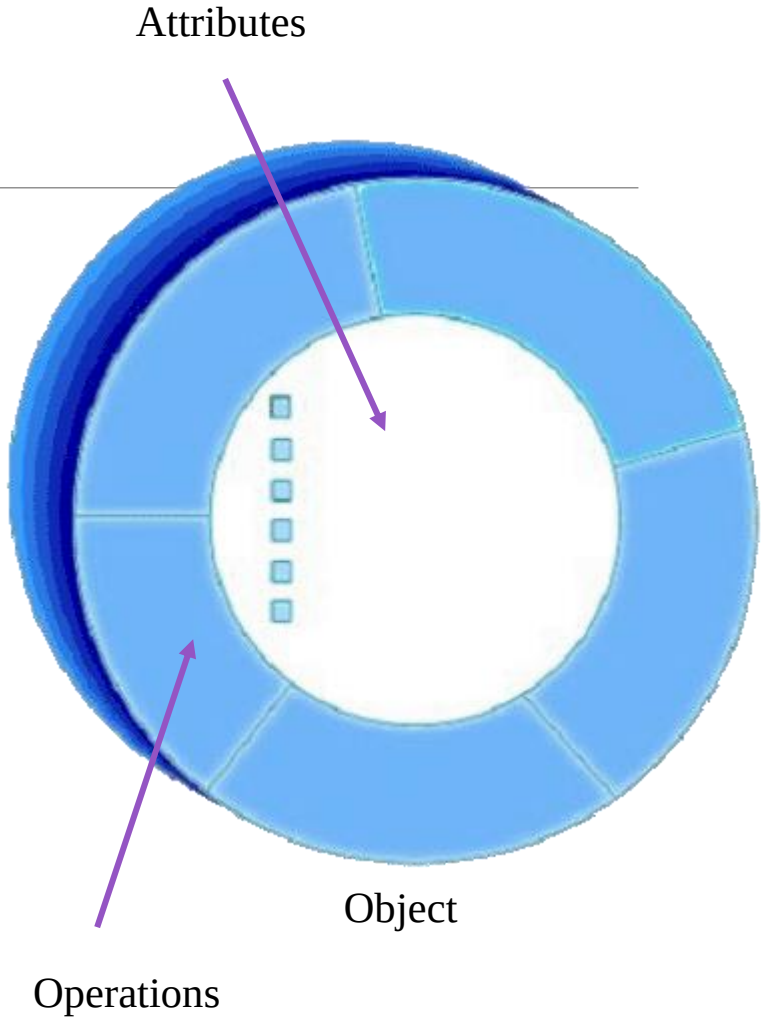


Linked List

A More Formal Definition

An object is an entity with a well-defined boundary and *identity* that encapsulates *state* and *behavior.*

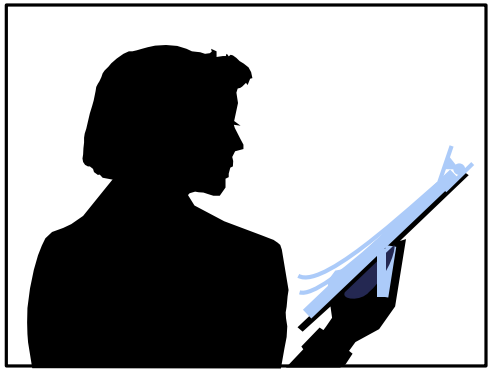
- State is represented by attributes and relationships.
- Behavior is represented by operations, methods, and state machines.



An Object Has State

State is a condition or situation during the life of an object, which satisfies some condition, performs some activity, or waits for some event.

The state of an object normally changes over time.



Name: J Clark
Employee ID: 567138
Date Hired: July 25, 1991
Status: Tenured
Discipline: Finance
Maximum Course Load: 3 classes

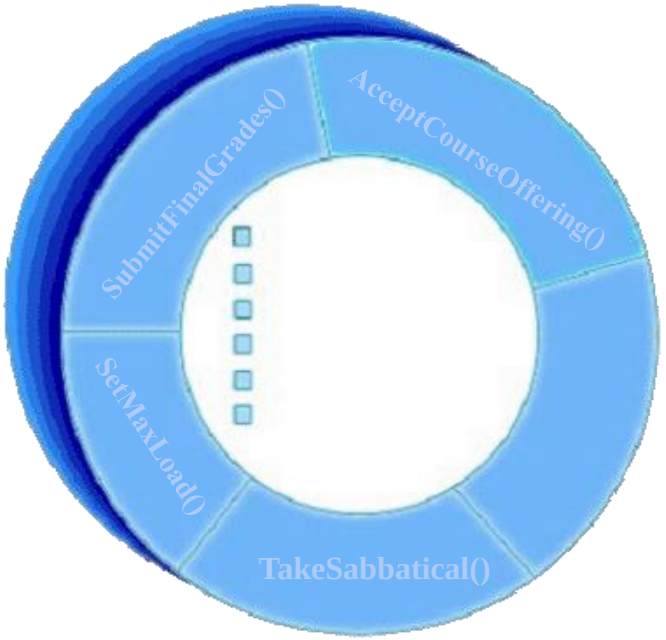
Professor Clark

An Object Has Behavior

Behavior determines how an object acts and reacts.

The visible behavior of an object is modeled by a set of messages it can respond to (operations that the object can perform).

Professor Clark's behavior
- Submit Final Grades
- Accept Course Offering
- Take Sabbatical
- Set Max Load



Professor Clark

An Object Has Identity

Each object has a unique identity, even if the state is identical to that of another object.

Professor “J Clark”
teaches Biology

Professor “J Clark”
teaches Biology

Basic Concept of Classes

Objects

Class

```
Mobile.java
1  package oopsconcept;
2
3  public class Mobile {
4      private String manufacturer;
5      private String operating_system;
6      public String model;
7      private int cost;
8
9      //Constructor to set properties/characteristics of object
10     Mobile(String man, String o,String m, int c){
11         this.manufacturer = man;
12         this.operating_system=o;
13         this.model=m;
14         this.cost=c;
15     }
16     //Method to get access Model property of Object
17     public String getModel(){
18         return this.model;
19     }
20     // We can add other method to get access to other properties
21 }
22
```

First Example: Using the Bicycle Class

```
class BicycleRegistration {
    public static void main(String[] args) {
        Bicycle bike1, bike2;
         String  owner1, owner2;

        bike1 = new Bicycle( );    //Create and assign values to bike1
        bike1.setOwnerName("Adam Smith");

        bike2 = new Bicycle( );    //Create and assign values to bike2
        bike2.setOwnerName("Ben Jones");

         owner1 = bike1.getOwnerName( ); //Output the information
         owner2 = bike2.getOwnerName( );

        System.out.println(owner1 + " owns a bicycle.");
        System.out.println(owner2 + " also owns a bicycle.");
    }
}
```

The Definition of the Bicycle Class

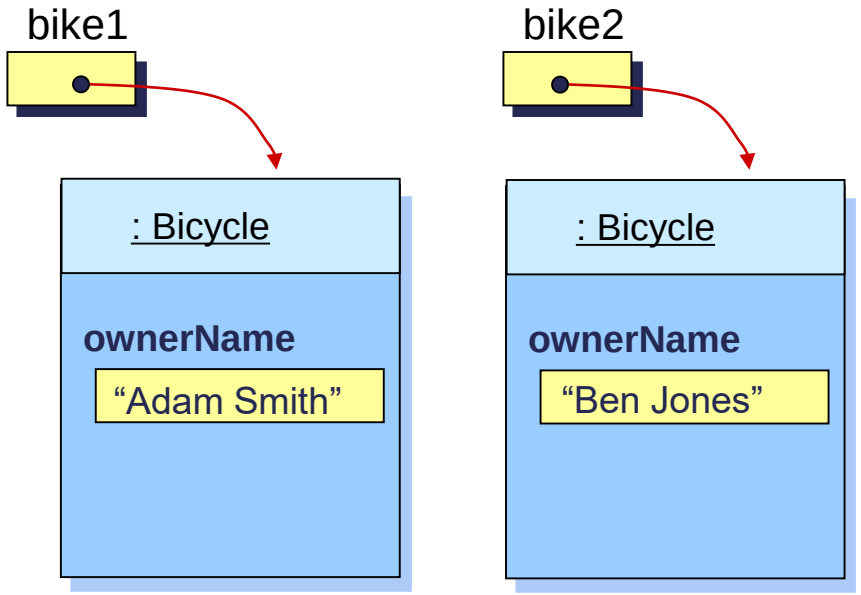
```
class Bicycle {

    // Data Member
    private String ownerName;

    //Constructor: Initialzes the data member
    public void Bicycle( ) {
        ownerName = "Unknown";
    }

    //Returns the name of this bicycle's owner
    public String getOwnerName( ) {

        return ownerName;
    }

    //Assigns the name of this bicycle's owner
    public void setOwnerName(String name) {

        ownerName = name;
    }
}
```

Multiple Instances

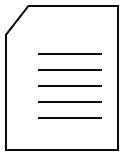
Once the Bicycle class is defined, we can create multiple instances.

```
Bicycle bike1, bike2;

bike1 = new Bicycle( );
bike1.setOwnerName("Adam Smith");

bike2 = new Bicycle( );
bike2.setOwnerName("Ben Jones");
```

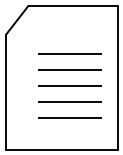
Sample Code



The Program Structure and Source Files



BicycleRegistration.java

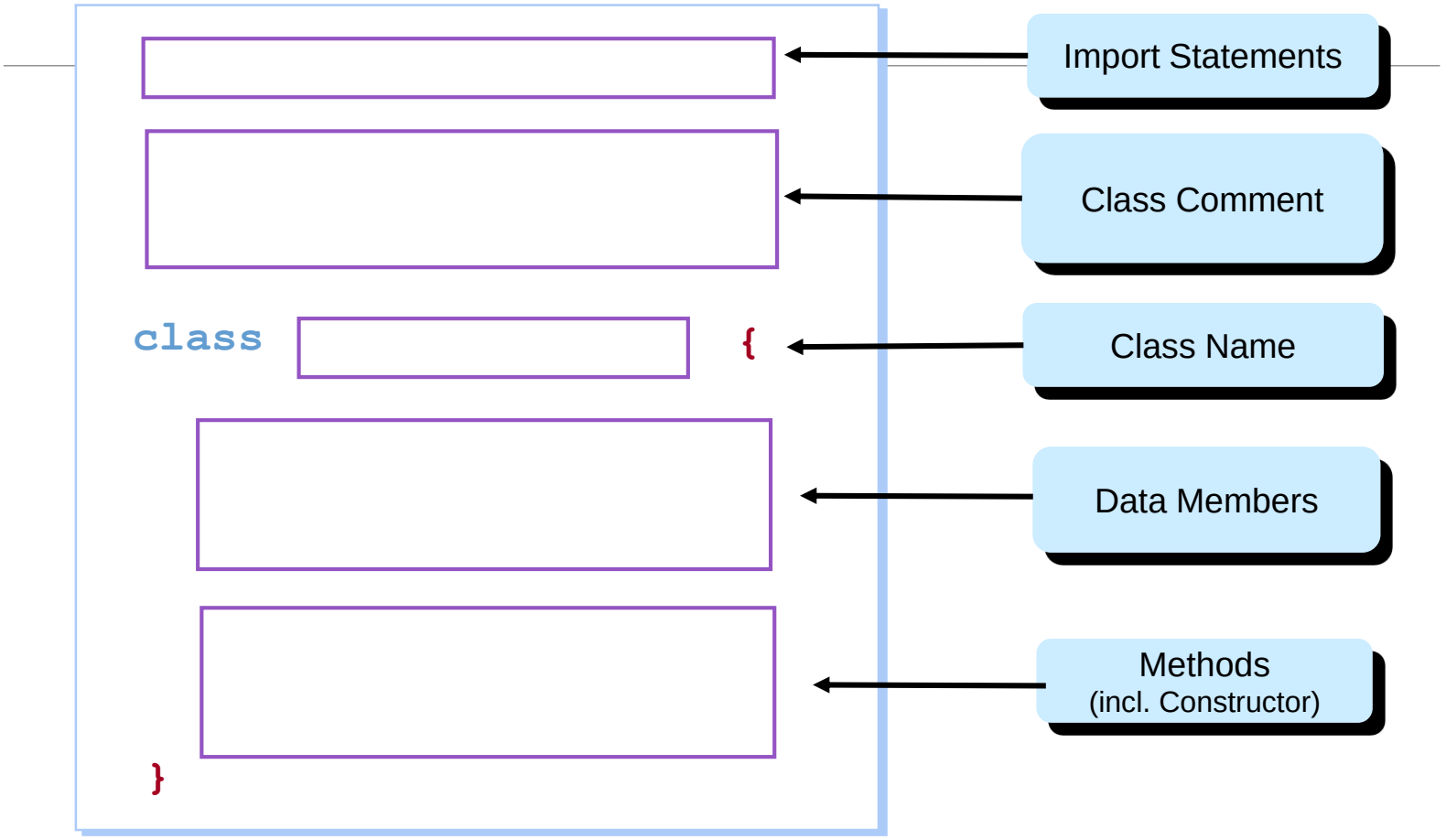


Bicycle.java

There are two source files. Each class definition is stored in a separate file.

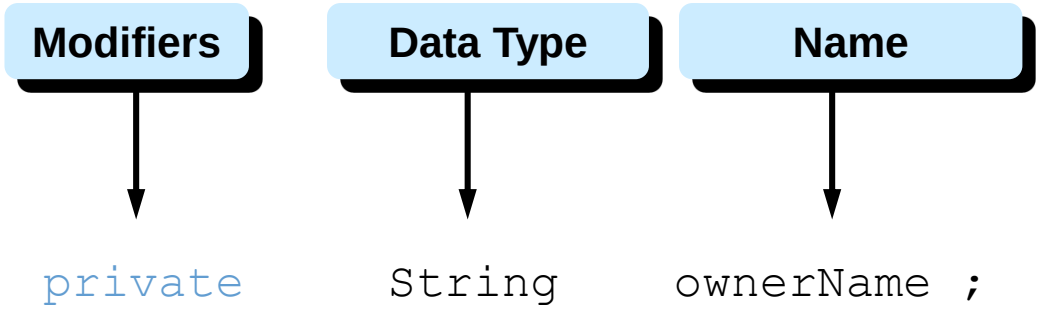
To run the program: 1. javac Bicycle.java (compile)
2. javac BicycleRegistration.java (compile)
3. java BicycleRegistration (run)

Template for Class Definition



Data Member Declaration

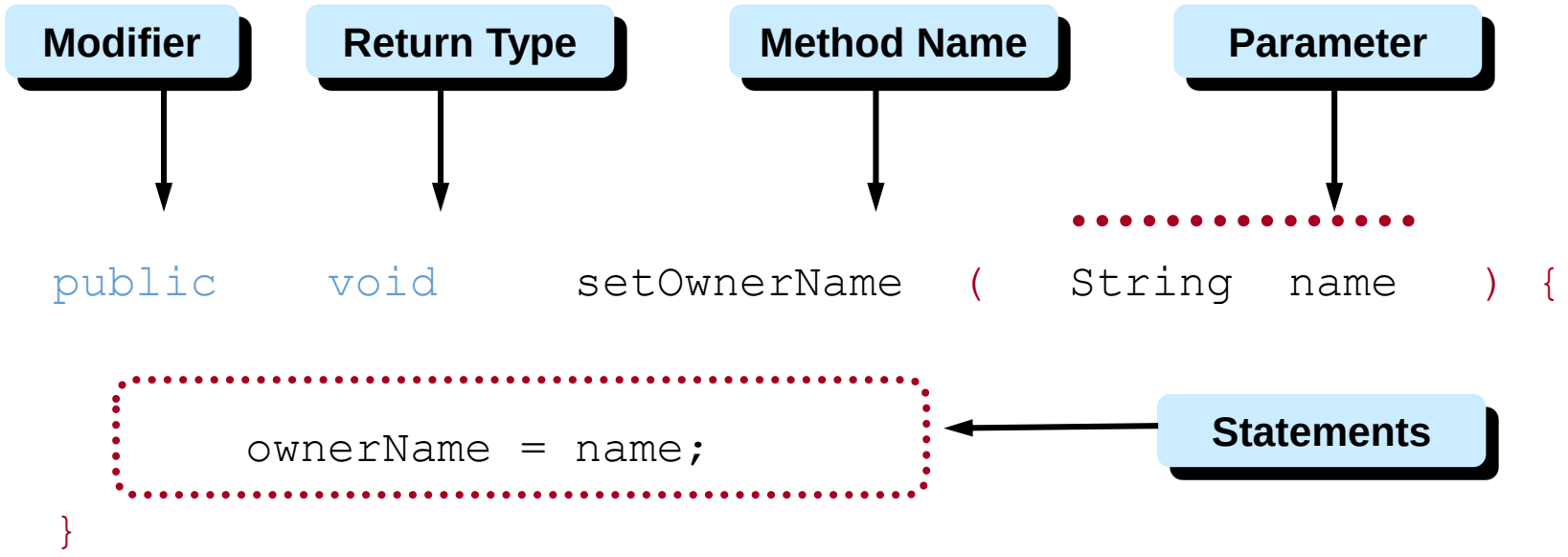
```
<modifiers>  <data type> <name> ;
```



Note: There's only one modifier in this example.

Method Declaration

```
<modifier>  <return type>  <method name>  ( <parameters>  ){
        <statements>
}
```



Quick Check!

1. Extend the Bicycle class by adding the second data member tagNo of String. Declare this data member as private.

2. Add new method to the Bicycle class that assigns a tag number. This method will be called as follows:

```
Bicycle bike;
bike = new Bicycle( );
...
bike.setTagNo("2004-134R");
```

Answers

1.

```
class Bicycle {

    // Data Members
    private String ownerName;
    private String tagNo;

   //Constructor: Initialzes the data member
   public void Bicycle( ) {
      ownerName = "Unknown";
      tagNo = "Unassigned";
   }

   ...
}
```

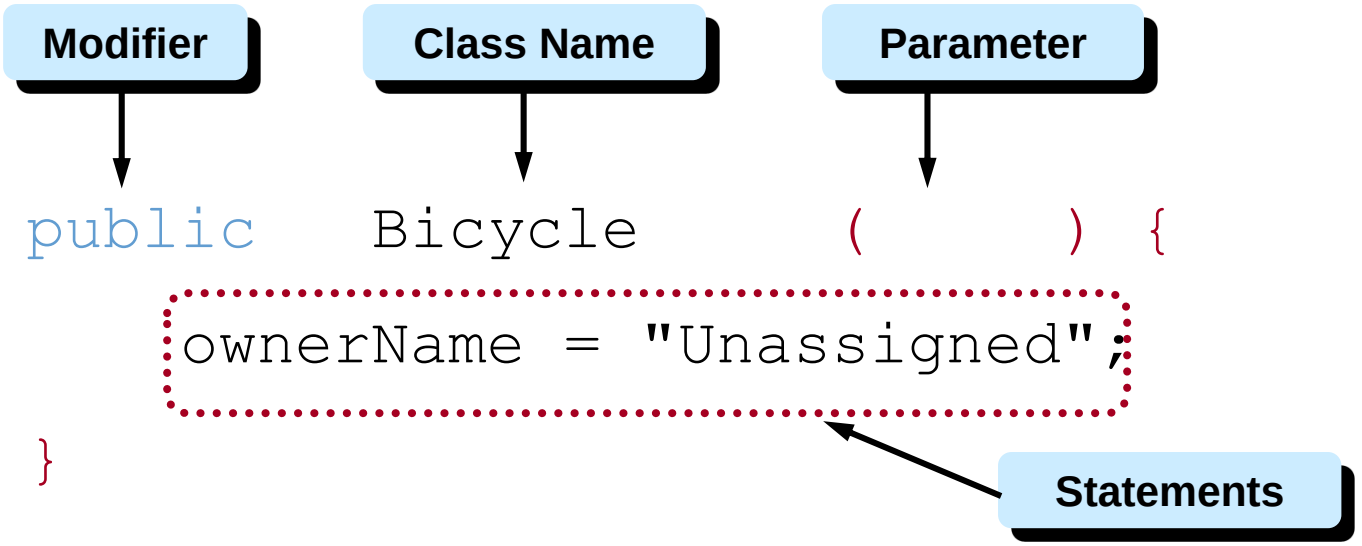
2.

```
class Bicycle {

    ...

    public void setTabNo(String number) {

        tagNo = number;
    }
    ...
}
```

Constructor

A *constructor* is a special method that is executed when a new instance of the class is created.

```
public <class name> ( <parameters> ){
    <statements>
}
```



Constructors

Type of constructor

- Default constructor (no argument)

```
public Bicycle() {
    gear = 1;
    cadence = 10;
    speed = 0;
}
```

```
public Bicycle() {

}
```

- Normal constructor (with argument)

```
public Bicycle(int startCadence, int startSpeed, int startGear)
{
    gear = startGear;
    cadence = startCadence;
    speed = startSpeed;
}
```

Constructors: Create new object

Default constructor (no argument)

```
public Bicycle() {
    gear = 1;
    cadence = 10;
    speed = 0;
}
```

Constructor

```
Bicycle yourBike = new Bicycle();
```

Create a new Bicycle object called yourBike

Constructors: Create new objects

Normal constructor (with argument)

```
public Bicycle(int startCadence, int startSpeed, int startGear)
{
    gear = startGear;
    cadence = startCadence;
    speed = startSpeed;
}
```

Constructor

```
Bicycle myBike = new Bicycle(30, 0, 8);
```

Create a new Bicycle object called myBike

Quick Check!

1. Which of the following constructors are invalid?

```
public int ClassA(int one) { ……..}

public ClassB(int one, int two) {……..}

void ClassC( ) {…….}
```

2. Complete the following constructor :

```
class Test {
private double score;
    public Test(double val) {
    //assign the value of parameter to
    //the data member
    }
}
```

Answers

1.

1. Invalid. The constructor does not have a return type.

2. Valid.

3. Invalid. The void *modifier is not valid with the constructor*

2.

```
class Test {
    private double score;

    public Test(double val) {
        score = val;
    }
}
```

Accessor (get method)

```
public String getOwnerName()
{
    return ownerName;
}
```

- Return information about an object
- getOwnerName()– return information about who is the owner of a bicycle

Mutator (set method)

//Assigns the name of this bicycle's owner

```
public void setOwnerName(String name)
 {
    ownerName = name;
 }
```

- Mutator - Sets a property(data,attrbute) of an object
- setOwnerName() is a mutator method

Second Example: Using Bicycle and Account

```
class SecondMain {

    //This sample program uses both the Bicycle and Account classes

    public static void main(String[] args) {

        Bicycle bike;
        Account acct;

        String  myName = "Jon Java";

        bike = new Bicycle( );
        bike.setOwnerName(myName);

        acct = new Account( );
        acct.setOwnerName(myName);
        acct.setInitialBalance(250.00);

        acct.add(25.00);
        acct.deduct(50);

        //Output some information
        System.out.println(bike.getOwnerName() + " owns a bicycle and");
        System.out.println("has $ " + acct.getCurrentBalance() +
                           " left in the bank");

    }
}
```

The Account Class

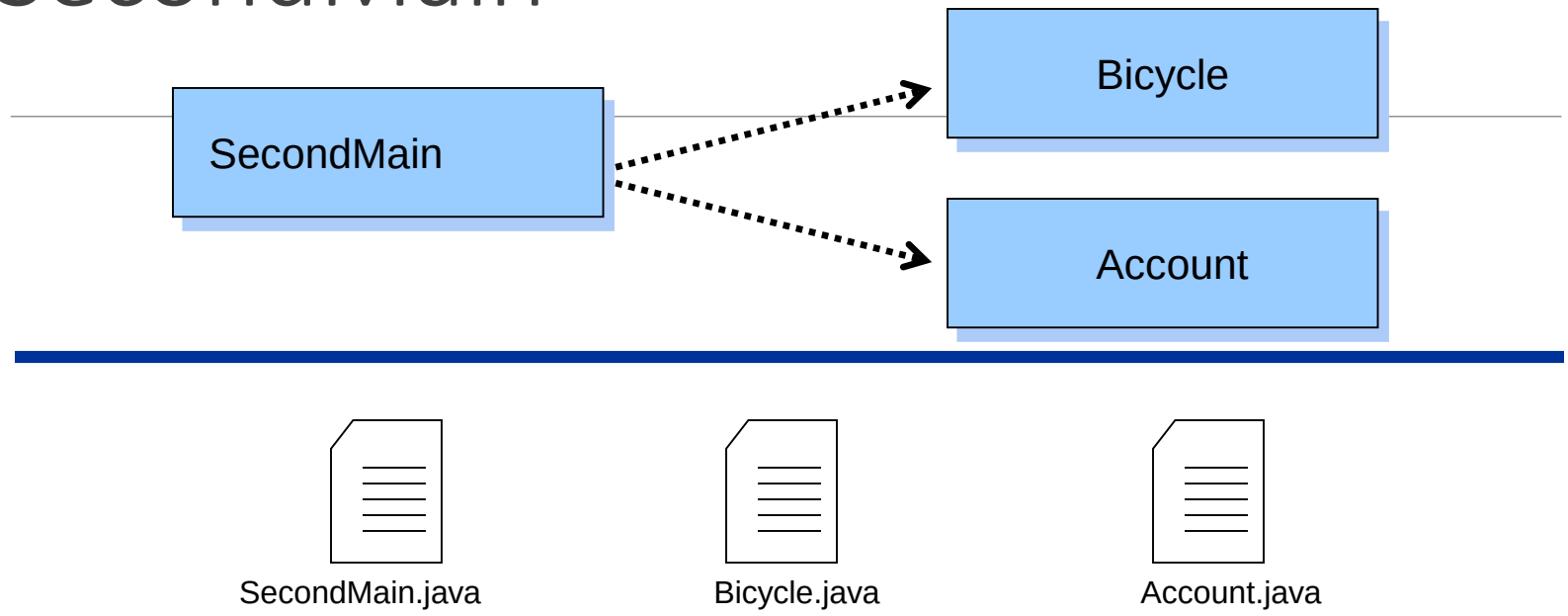
```
class Account {

    private String ownerName;

    private double balance;

    public Account( ) {
         ownerName = "Unassigned";
         balance = 0.0;
    }

    public void add(double amt) {
        balance = balance + amt;
    }

    public void deduct(double amt) {
        balance = balance - amt;
    }

    public double getCurrentBalance( ) {
        return balance;
    }

    public String getOwnerName( ) {

        return ownerName;
    }
```

Page 1

```
    public void setInitialBalance
                            (double bal) {

         balance = bal;
    }

    public void setOwnerName
                          (String name) {

        ownerName = name;
    }
}
```

Page 2

The Program Structure for SecondMain



To run the program:
1. javac Bicycle.java (compile)
2. javac Account.java (compile)
2. javac SecondMain.java (compile)
3. java SecondMain (run)

Note: You only need to compile the class once. Recompile only when you made changes in the code.

Quick Check!

- What is the output from the following code fragment?

```
Account acct;
acct = new Account( );
acct.setInitialBalance(250);
acct.add(20);
System.out.println("Balance: " +
acct.getCurrentBalance());
```

- Write a code fragment to declare and create two Account objects named acc1 and acct2. Initialize the balance to $300 and $500, respectively.

Answers

1.

```
Balance: 270
```

2.

```
Account acct1, acct2;
acct1 = new Account( );
acct2 = new Account( );

acct1.setInitialBalance(300);
acct2.setInitialBalance(300);
```

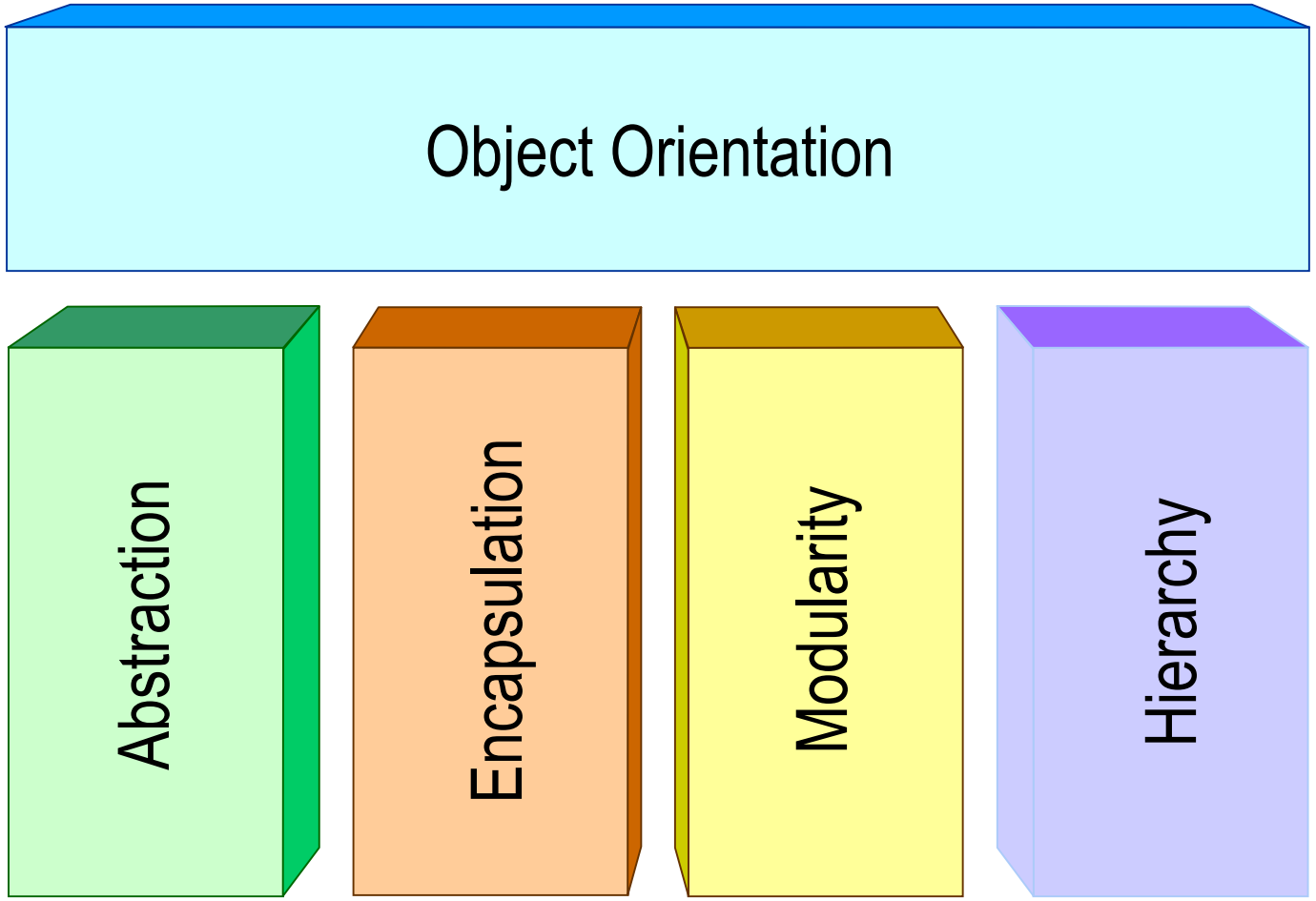
Discussions: Design of Home Automation System

Jamie and Kelly own a digital shower. They can give instructions to the shower from any room or even via their smartphones. The home automation system has state-of-the-art technology for voice and face recognition, so can recognize that it's Jamie when he calls in at 3.00 am on the way home from a hackathon and says: "gimme a hot shower in 5 minutes". It knows that "hot shower" for Jamie is a different temperature than a "hot shower" for Kelly. The system can play appropriate music in the shower, and knows what are Kelly's current hot favourites on her MP3 Player.

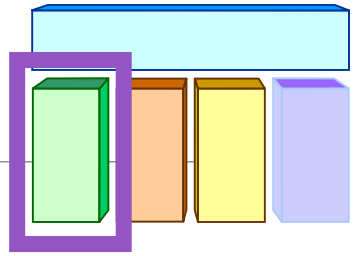
Here are some questions:

1. What are the important classes that the home automation system needs to know about?
2. What methods will objects of those classes need to make available to the system?

Basic Principles of Object Orientation



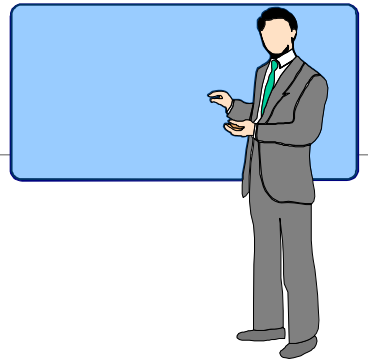
What Is Abstraction?



- The essential characteristics of an entity that distinguishes it from all other kinds of entities.
- Defines a boundary relative to the perspective of the viewer.
- Is not a concrete manifestation, denotes the ideal essence of something.

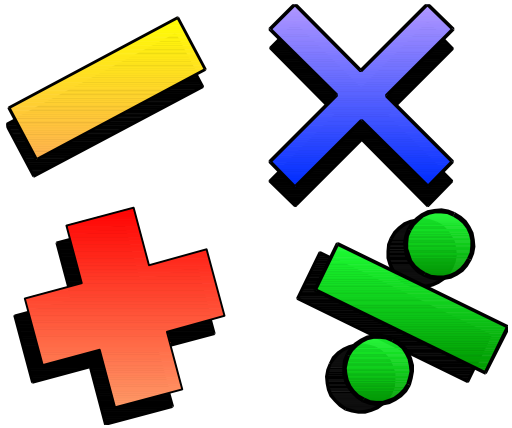
Example: Abstraction

Student



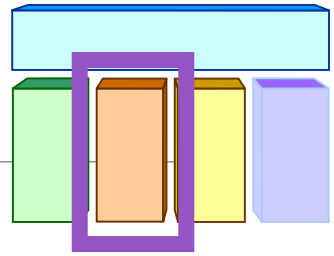
Professor

Course Offering (9:00 a.m., Monday-Wednesday-Friday)

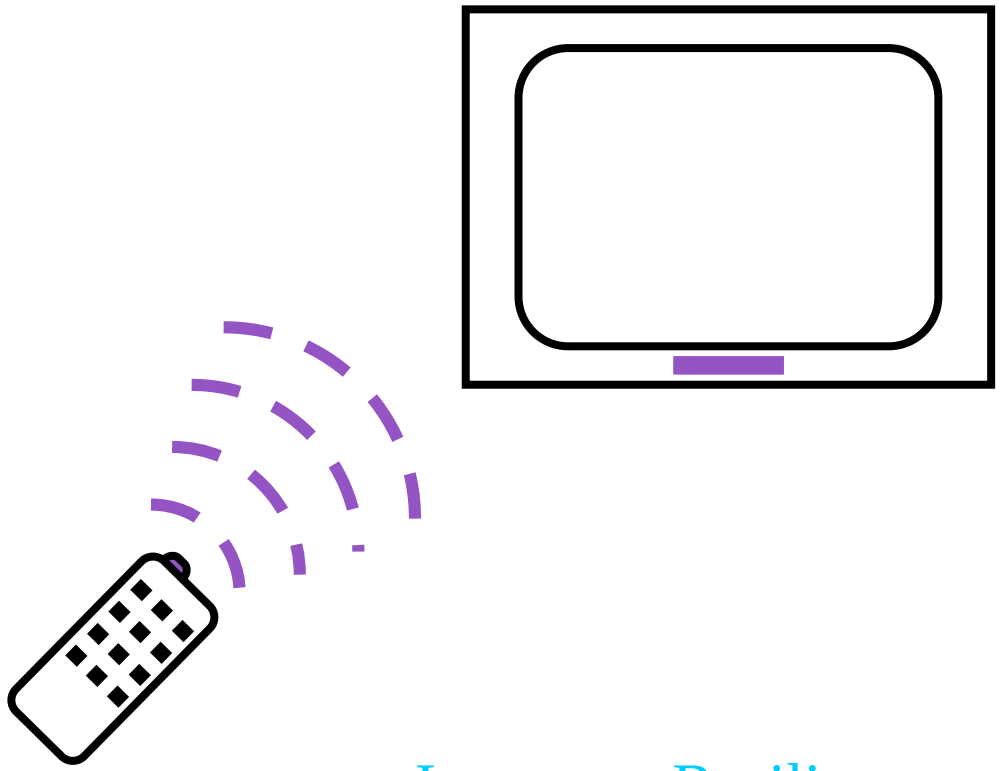


Course (e.g. Algebra)

What Is Encapsulation?



- Hides implementation from clients.
  - Clients depend on interface.



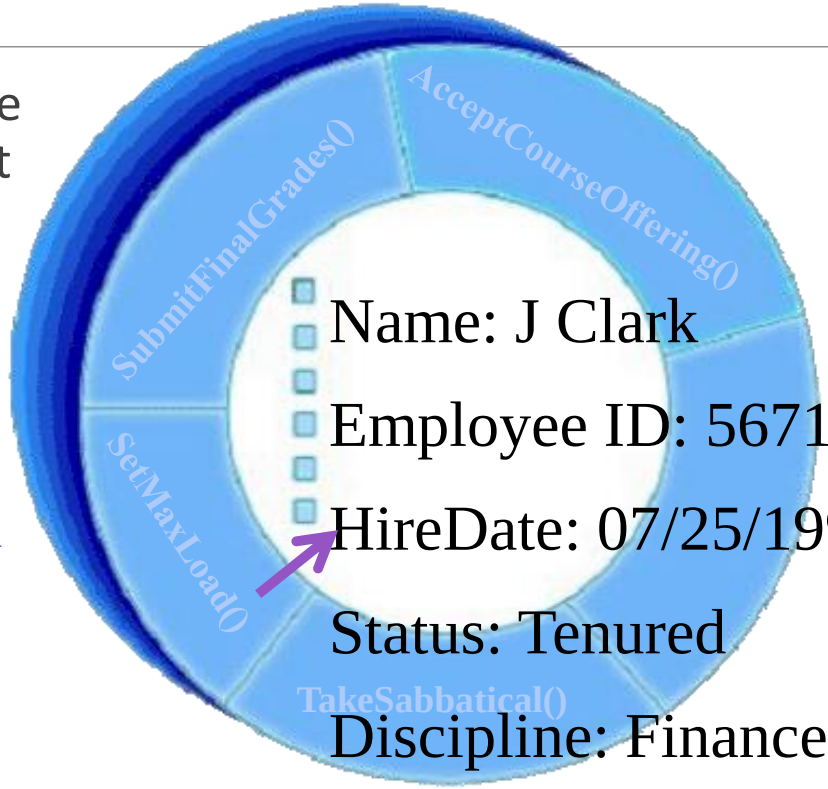
Improves Resiliency

Encapsulation Illustrated

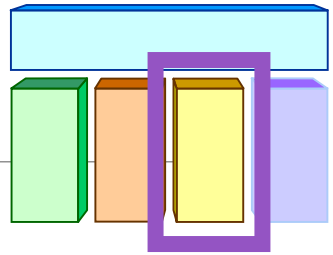
Professor Clark

Professor Clark needs to be able to teach four classes in the next semester.

SetMaxLoad(4)

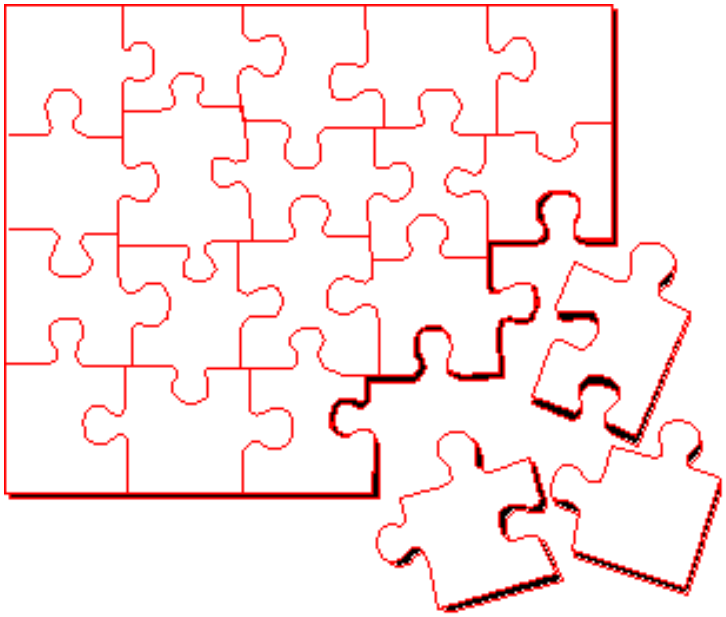


What Is Modularity?



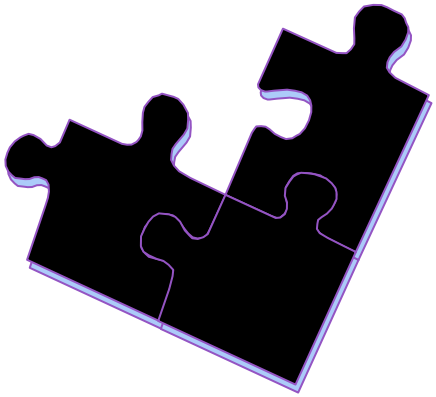
Breaks up something complex into manageable pieces.

Helps people understand complex systems.

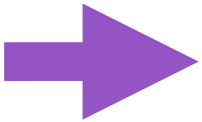


Example: Modularity

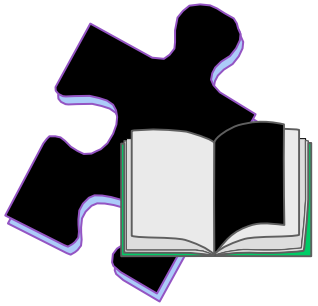
For example, break complex systems into smaller modules.



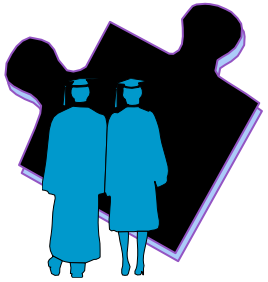
Course Registration System



Billing System

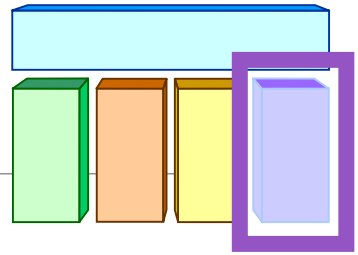


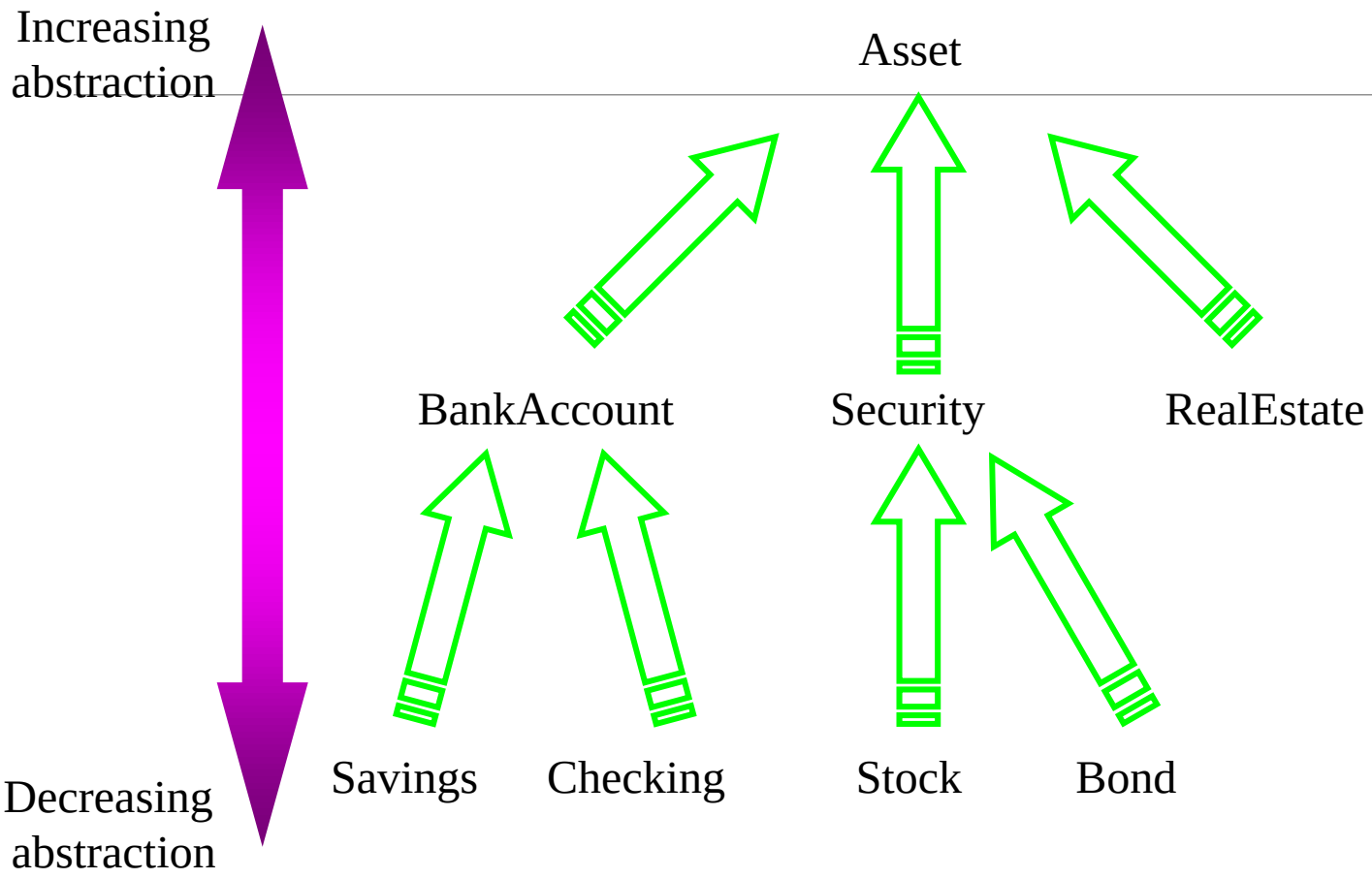
Course Catalog System



Student Management System

What Is Hierarchy?

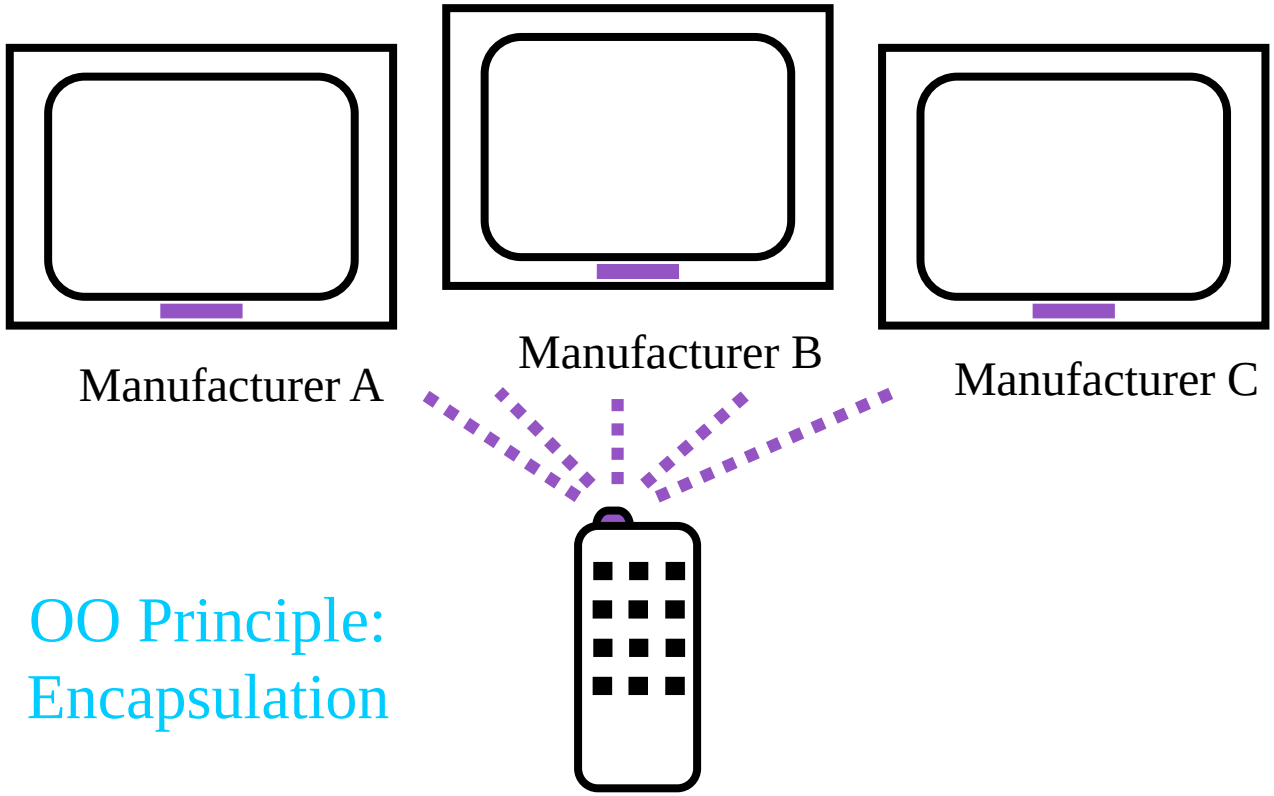




Elements at the same level of the hierarchy should be at the same level of abstraction.

What Is Polymorphism?

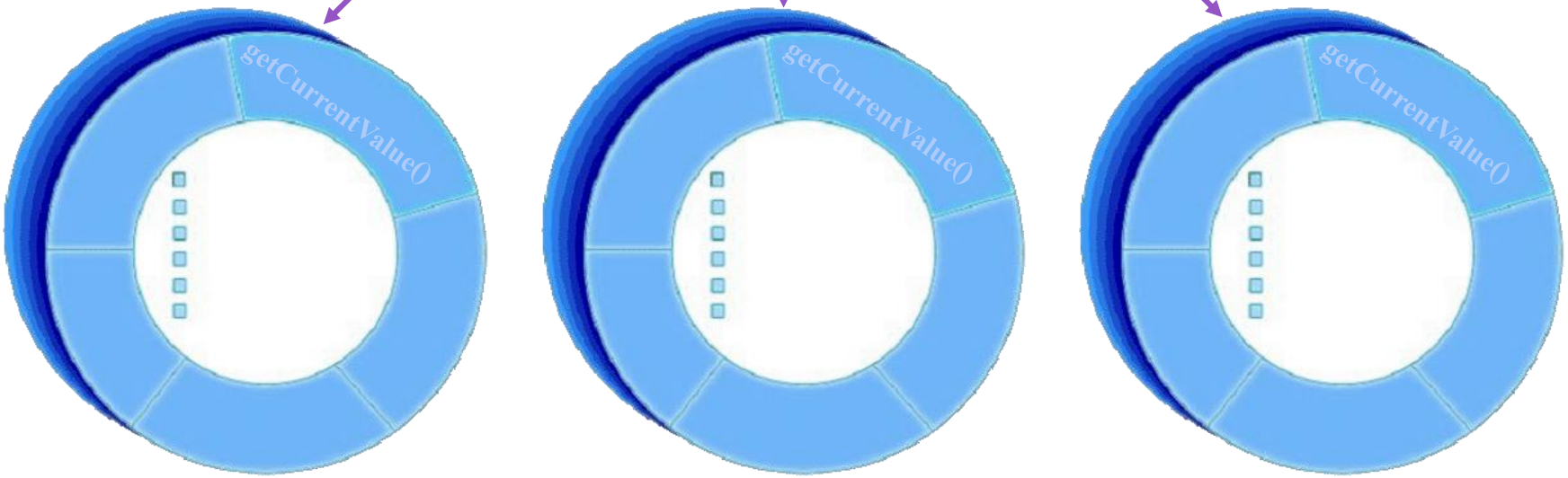
- The ability to hide many different implementations behind a single interface.



OO Principle:
Encapsulation

Example: Polymorphism

financialInstrument.getCurrentValue()



Stock Bond Mutual Fund

What Is Generalization?

A relationship among classes where one class shares the structure and/or behavior of one or more classes.

Defines a hierarchy of abstractions in which a subclass inherits from one or more superclasses.

- **Single inheritance.**
- **Multiple inheritance.**

Is an “is a kind of” relationship.

Tutorial 1

Suppose that we want to define the class Waterbill to calculate the usage of water for a month. The bill represented the integer for the account number and double for water usage and monthly bill. Write WaterBill class with the following tasks:

- Get water usage for a month
- Calculate the bill for a month
- Display account number, water bill and water usage for a month