Objects and Classes

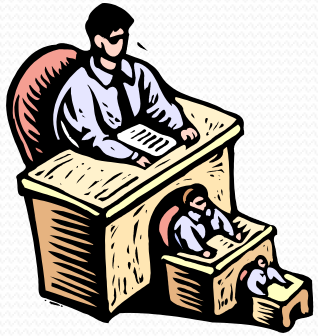
Session 3

The Object Oriented Approach -I

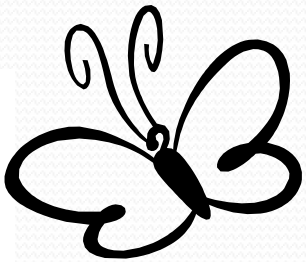
- Object oriented programming grew in the 70's as a solution to the problems of structured programming
- Models human thought process as closely as possible
- Deals with data and procedures that operate on data as a single 'object'

The Object Oriented Approach - II

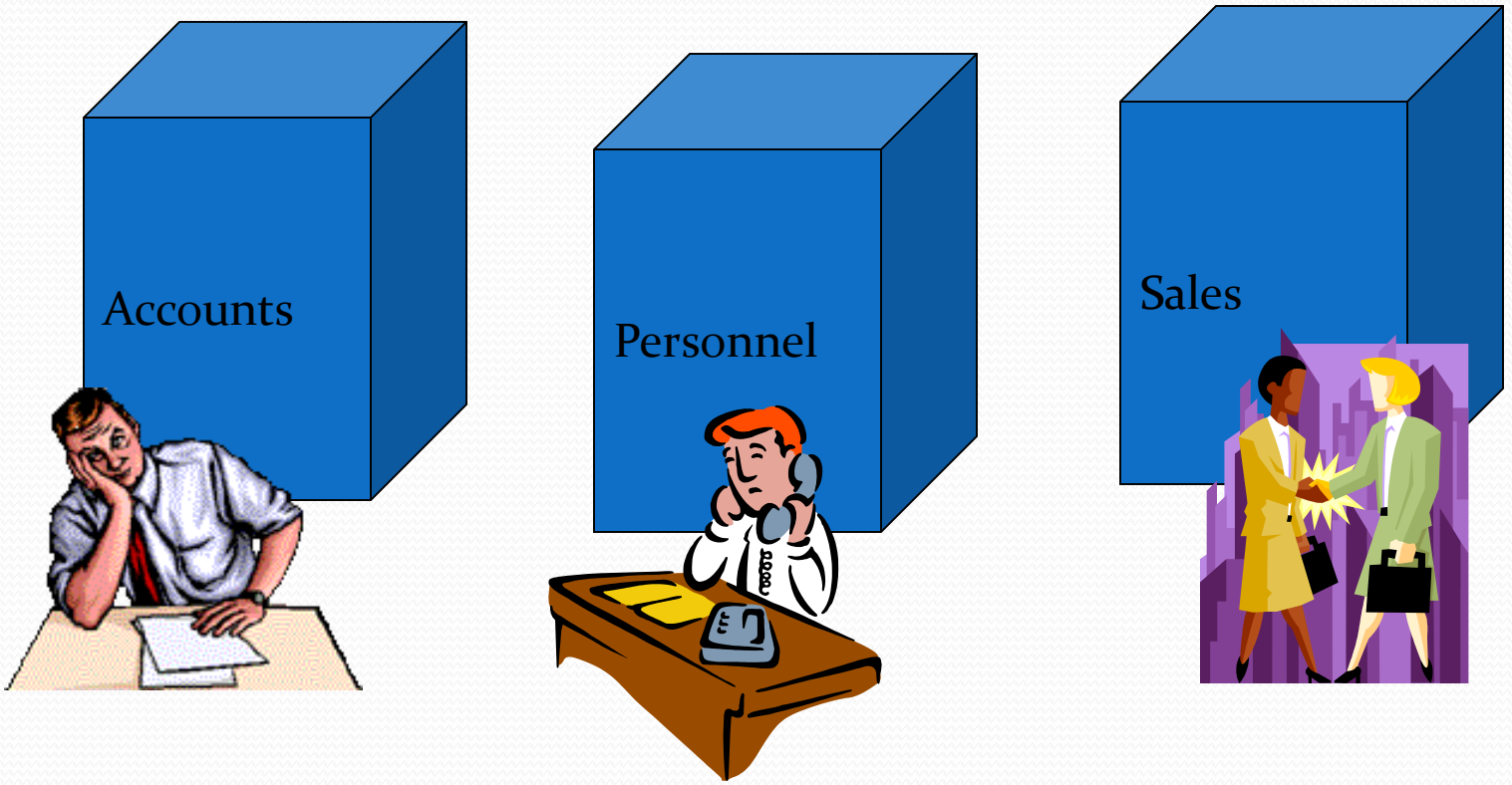
All around us in the real world are objects.



Each object has certain characteristics and exhibits certain behaviour



The Object-Oriented Approach - III



The real world around is full of objects .We can consider both living beings as well as things as objects.For example,the different departments in a company are objects.

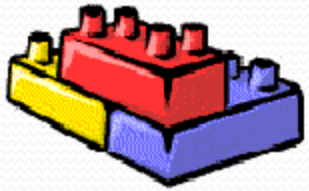
Drawbacks of Traditional Programming

The drawbacks of Traditional Programming are:

- Unmanageable programs
- Problems in modification of data
- Difficulty in implementation

Why do we care about objects?

- Modularity - large software projects can be split up in smaller pieces.
- Reuseability - Programs can be assembled from pre-written software components.
- Extensibility - New software components can be written or developed from existing ones.



Object – Oriented Programming

Here the application has to implement the entities as they are seen in real life and associate actions and attributes with each.

Data

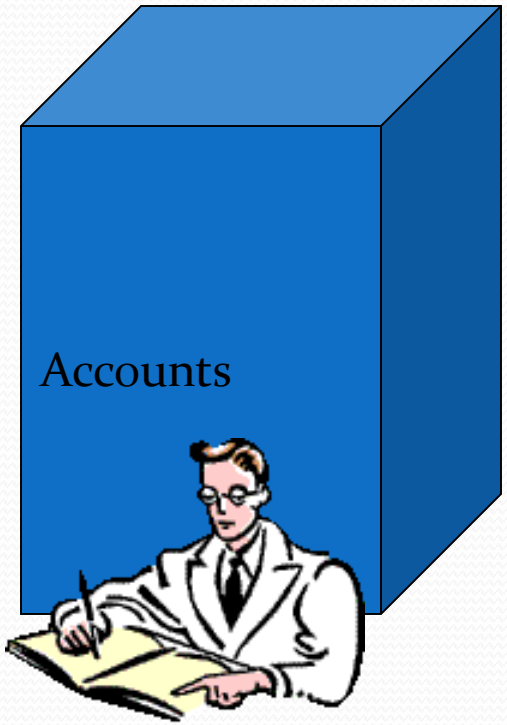
Employee details

Salary statements

Bills

Vouchers

Reciepts



Functions

Calculate salary

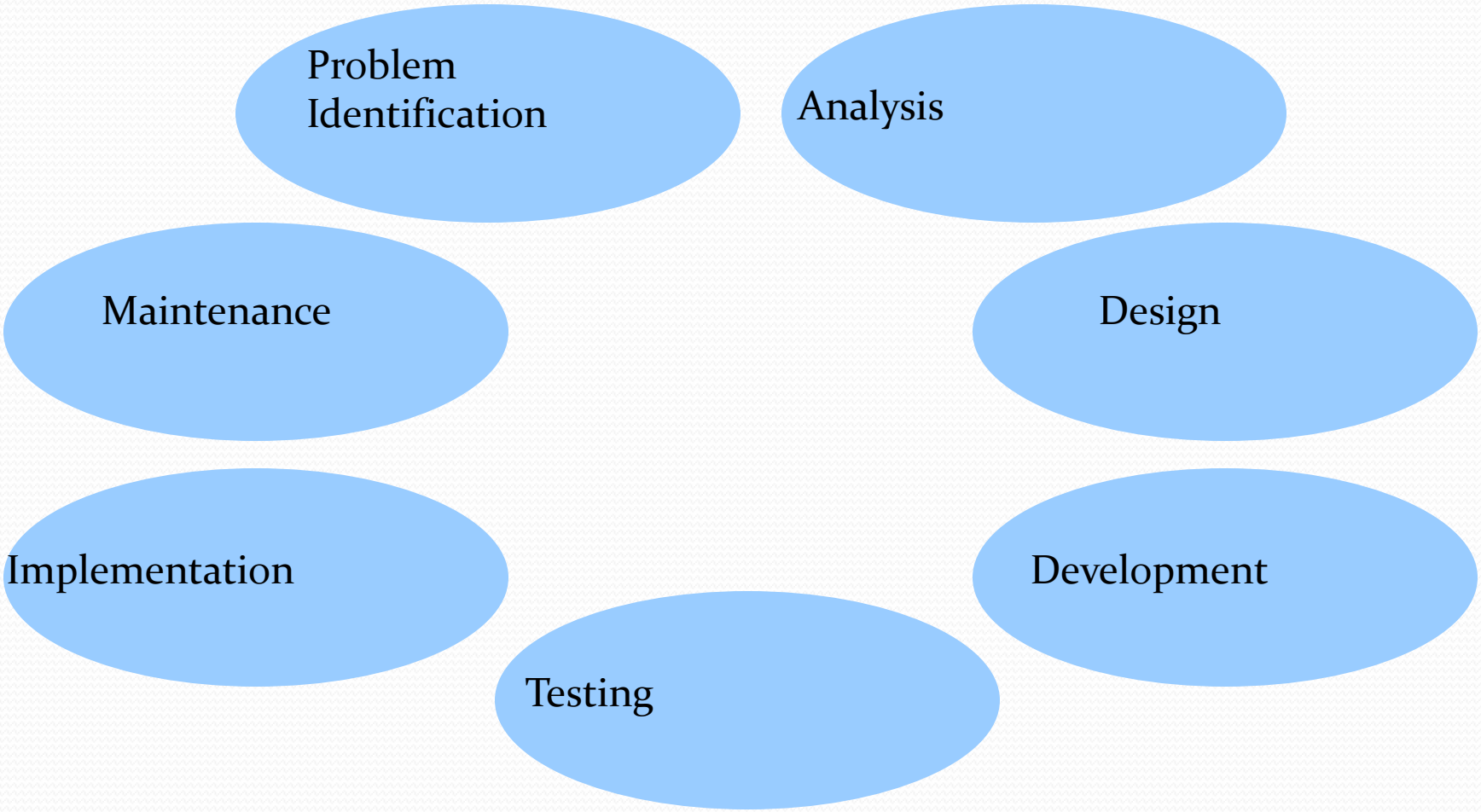
Pay salary

Pay bills

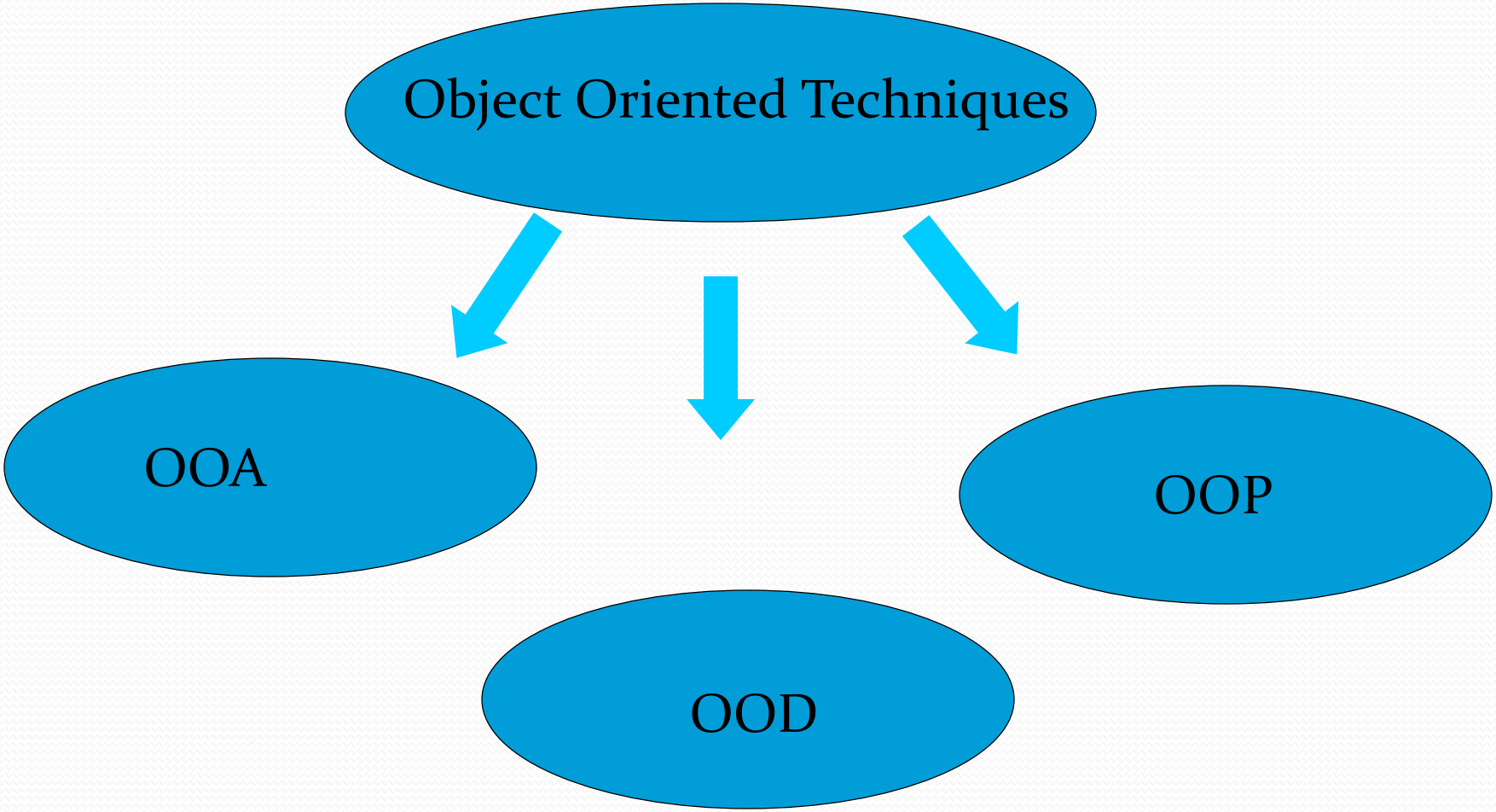
Tally accounts

Transact with banks

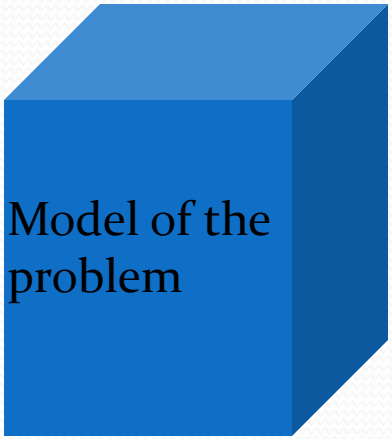
Object Oriented Approach

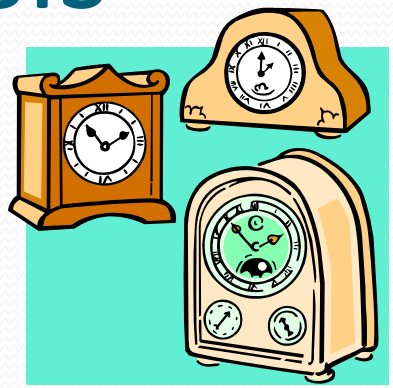


Object Oriented Techniques



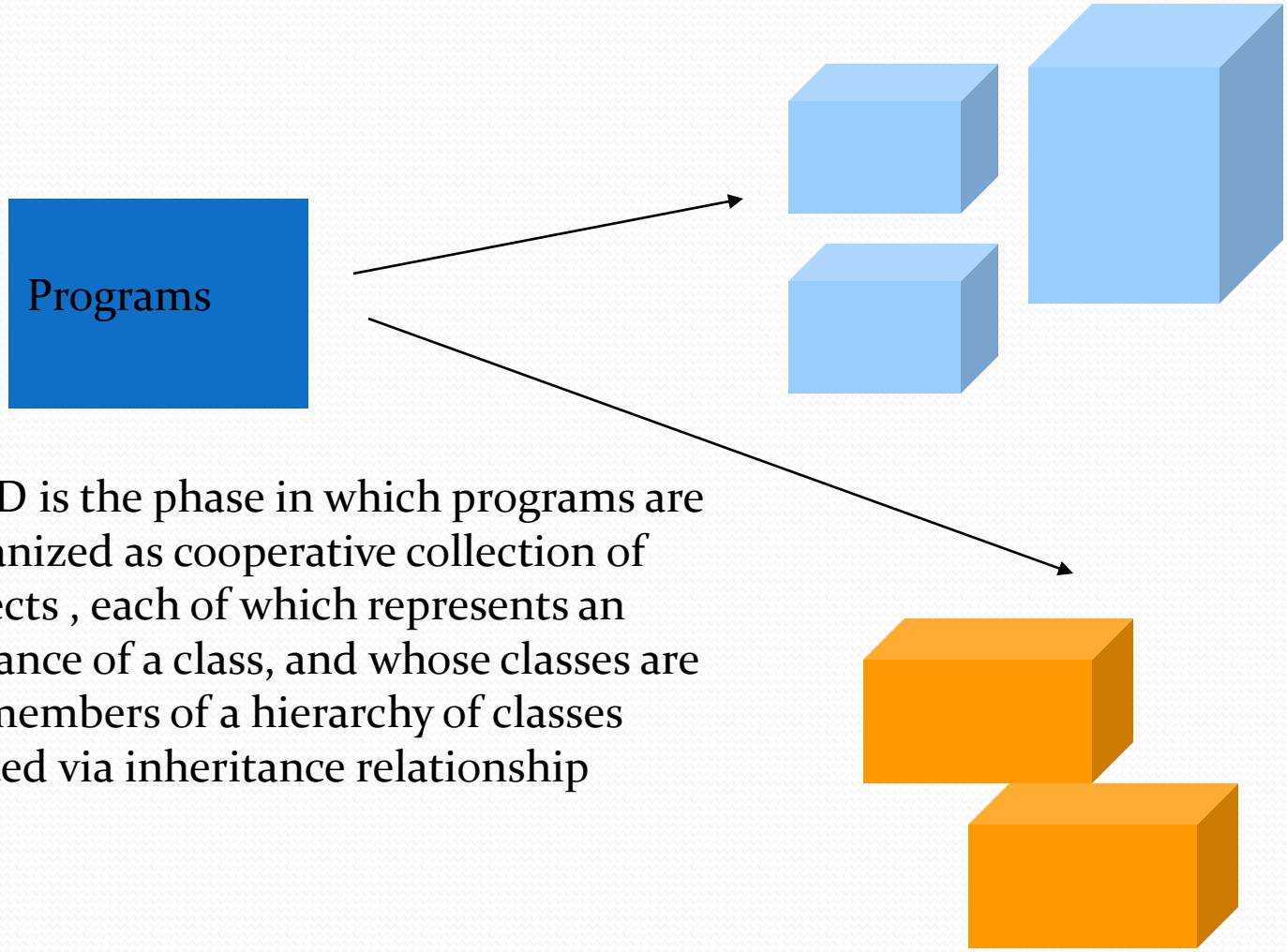
Object Oriented Analysis





OOA is the phase if any project during which a precise and concise model of the problem in terms of real world objects and concepts as understood by the user is developed

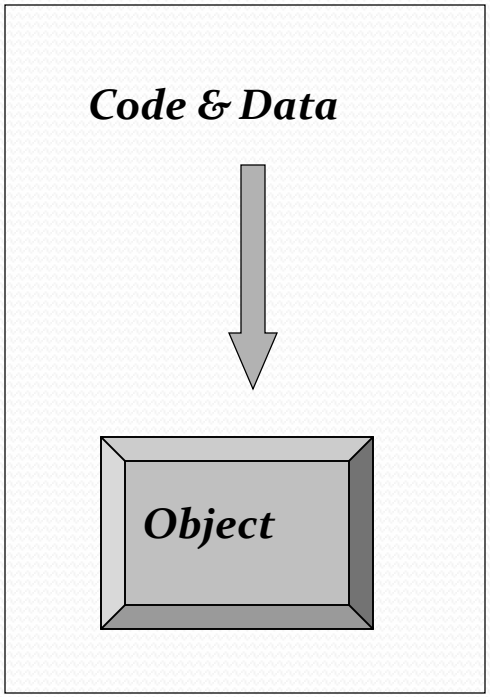
Object Oriented Design



OOD is the phase in which programs are organized as cooperative collection of objects , each of which represents an instance of a class, and whose classes are all members of a hierarchy of classes united via inheritance relationship

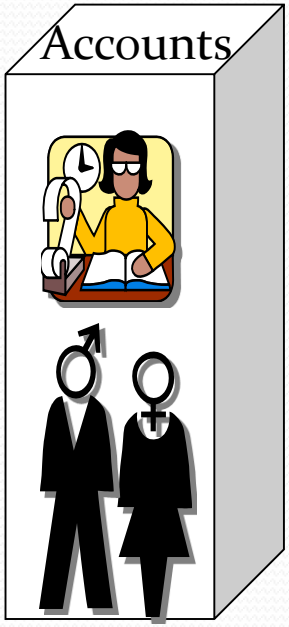
Object Oriented Programming

OOP (Object oriented Programming) is the construction phase of the life cycle that Object-Oriented Techniques follows



Data:

- *No. of employees*
- *Salary statements*
- *Bills*
- *Vouchers*
- *Receipts*
- *Petty cash records*
- *Banking data*



Functions:

- *Calculate salary*
- *Pay salary*
- *Pay bills*
- *Tally accounts*
- *Transact with banks*

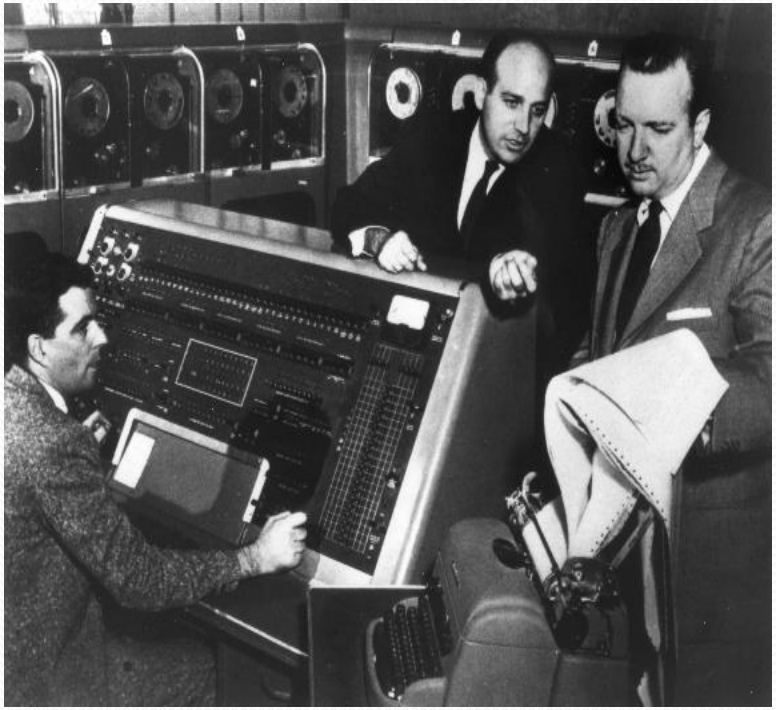
Basic Object Oriented Concepts

- Object
  - Helps to understand the real world
  - Provides a practical basis for computer applications
- Class
  - Describes a set of related objects
- Property
  - A characteristic of an object – also called *attribute*
- Method
  - An action performed by an object

Object-Oriented Programming

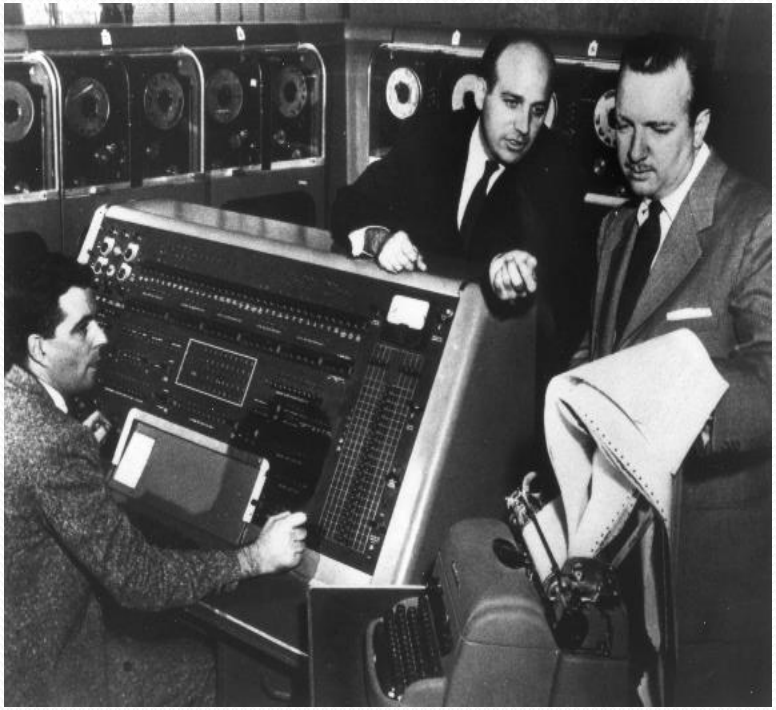
Early computers were far less complex than computers are today.

Their memories were smaller and their programs were much simpler.



Object-Oriented Programming

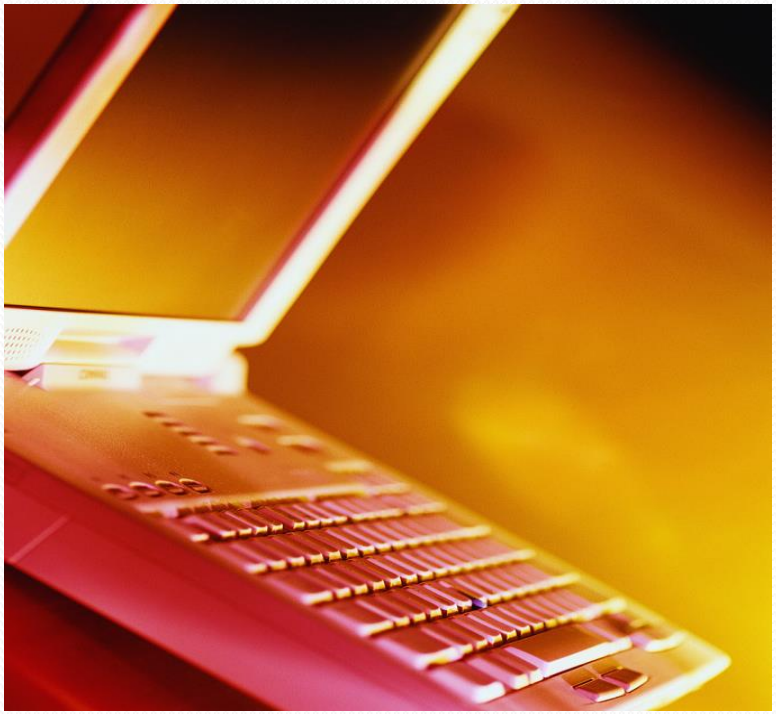
They usually executed only one program at a time.



Object-Oriented Programming

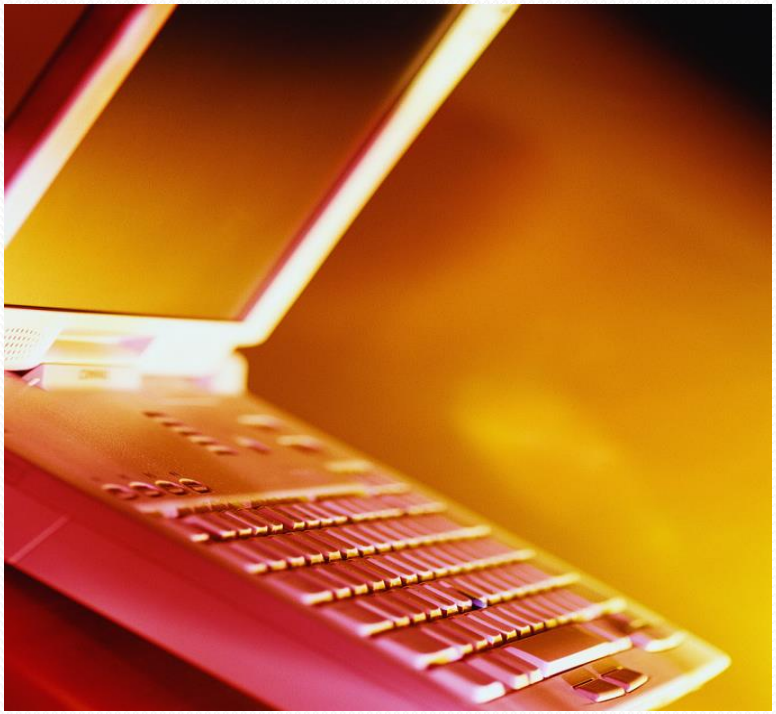
Modern computers are smaller, but far more complex than early computers.

They can execute many programs at the same time.



Object-Oriented Programming

Computer scientists have introduced the notion of ***objects*** and ***object-oriented programming*** to help manage the growing complexity of modern computers.



Object-Oriented Programming

An ***object*** is anything that can be represented by data in a computer's memory and manipulated by a computer program.

Object-Oriented Programming

An ***object*** is anything that can be represented by data in a computer's memory and manipulated by a computer program.

Numbers

Object-Oriented Programming

An ***object*** is anything that can be represented by data in a computer's memory and manipulated by a computer program.

Text

Object-Oriented Programming

An ***object*** is anything that can be represented by data in a computer's memory and manipulated by a computer program.

Pictures

Object-Oriented Programming

An ***object*** is anything that can be represented by data in a computer's memory and manipulated by a computer program.

Sound

Object-Oriented Programming

An ***object*** is anything that can be represented by data in a computer's memory and manipulated by a computer program.

Video

Object-Oriented Programming

An object is anything that can be represented by data.

Object-Oriented Programming

An object can be something in the physical world or even just an abstract idea.

An airplane, for example, is a physical object that can be manipulated by a computer.

Object-Oriented Programming

An object can be something in the physical world or even just an abstract idea.

A bank transaction is an example of an object that is not physical.

Object-Oriented Programming

To a computer, an object is simply something that can be represented by data in the computer's memory and manipulated by computer programs.

Object-Oriented Programming

The data that represent the object are organized into a set of ***properties***.

The values stored in an object's properties at any one time form the ***state*** of an object.

Name:	PA 3794
Owner:	Pakistan International Airline
Location:	39 52′ 06″ N 75 13′ 52″ W
Heading:	271°
Altitude:	19 m
AirSpeed:	0
Make:	Boeing
Model:	737
Weight:	32,820 kg

Fields – Declaration

- **Field Declaration**
  - a type name followed by the field name, and optionally an initialization clause
  - primitive data type vs. Object reference
    - boolean, char, byte, short, int, long, float, double
  - field declarations can be preceded by different modifiers
    - access control modifiers
    - static
    - final

More about field modifiers (1)

- Access control modifiers
  - *private*: private members are accessible only in the class itself
  - *package*: package members are accessible in classes in the same package and the class itself
  - *protected*: protected members are accessible in classes in the same package, in subclasses of the class, and in the class itself
  - *public*: public members are accessible anywhere the class is accessible

Pencil.java

```
public class Pencil {
      public String color = "red";
      public int length;
      public float diameter;
      private float price;

      public static long nextID = 0;

      public void setPrice (float newPrice) {
            price = newPrice;
      }
 }
```

CreatePencil.java

```
public class CreatePencil {
      public static void main (String args[]){
                Pencil p1 = new Pencil();
                p1.price = 0.5f;
      }
}
```

```
%> javac Pencil.java
%> javac CreatePencil.java
CreatePencil.java:4: price has private access in Pencil
        p1.price = 0.5f;
           ^
```

More about field modifiers (2)

- static
  - only one copy of the static field exists, shared by all objects of this class
  - can be accessed directly in the class itself
  - access from outside the class must be preceded by the class name as follows

```
System.out.println(Pencil.nextID);
```

    or via an object belonging to the class
  - from outside the class, non-static fields must be accessed through an object reference

```
public class CreatePencil {
      public static void main (String args[]){
              Pencil p1 = new Pencil();
              Pencil.nextID++;
              System.out.println(p1.nextID);
              //Result?   1

              Pencil p2 = new Pencil();
              Pencil.nextID++;
              System.out.println(p2.nextID);
              //Result?

              System.out.println(p1.nextID);
              //Result?    2
      }
}
```

still 2!

Note: this code is only for the purpose of showing the usage of static fields. It has POOR design!

More about field modifiers (3)

- final
  - once initialized, the value cannot be changed
  - often be used to define named constants
  - static final fields must be initialized when the class is initialized
  - non-static final fields must be initialized when an object of the class is constructed

Object-Oriented Programming

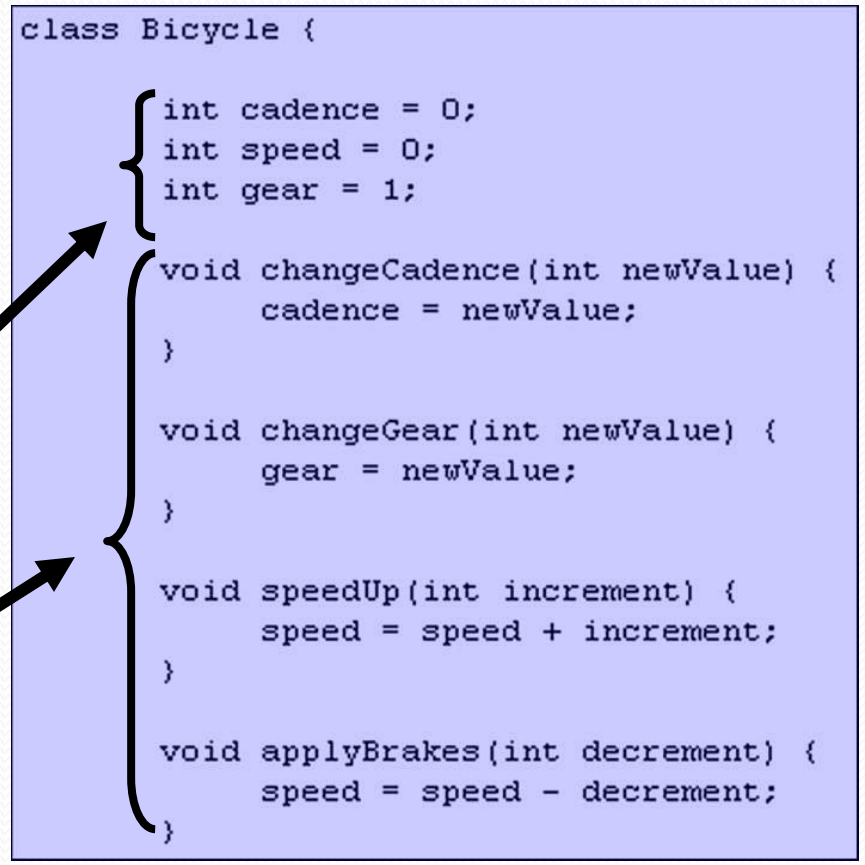
Computer programs implement algorithms that manipulate the data.

In object-oriented programming, the programs that manipulate the properties of an object are the object's **methods**.

```
class Bicycle {

        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;
        }
```

Object-Oriented Programming

We can think of an object as a collection of properties and the methods that are used to manipulate those properties.



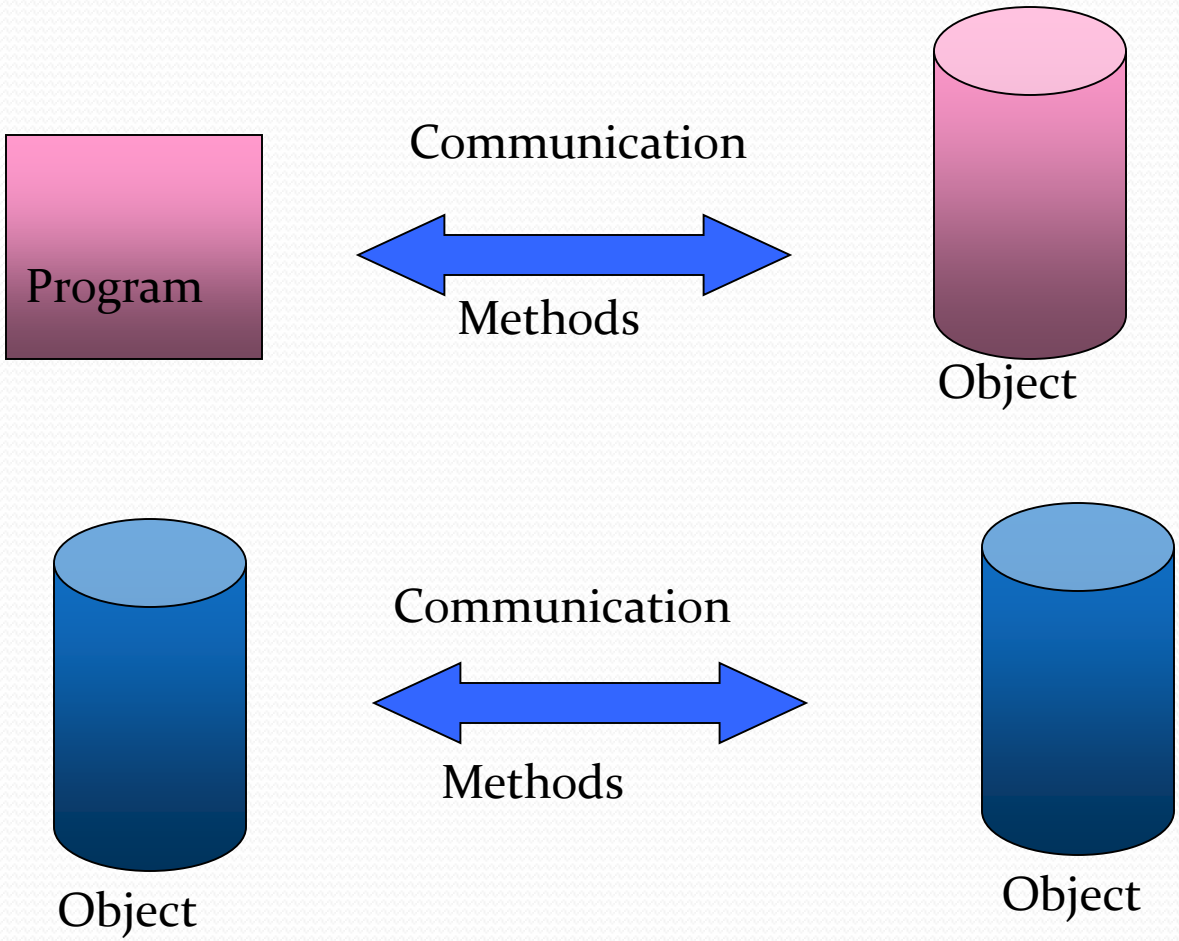
Methods – Declaration

- Method declaration: two parts
  1. method header
     - consists of modifiers (optional), return type, method name, parameter list and a throws clause (optional)
     - types of modifiers
       - *access control modifiers*
       - *abstract*
         - the method body is empty. E.g.

```
abstract void sampleMethod( );
```

       - *static*
         - represent the whole class, no a specific object
         - can only access static fields and other static methods of the same class
       - *final*
         - cannot be overridden in subclasses
  2. method body

Calling Methods



Methods – Invocation

- Method invocations
  - invoked as operations on objects/classes using the dot (.) operator

```
reference.method(arguments)
```

  - static method:
    - Outside of the class: "`reference`" can either be the class name or an object reference belonging to the class
    - Inside the class: "`reference`" can be ommitted
  - non-static method:
    - "`reference`" must be an object reference

Calling methods

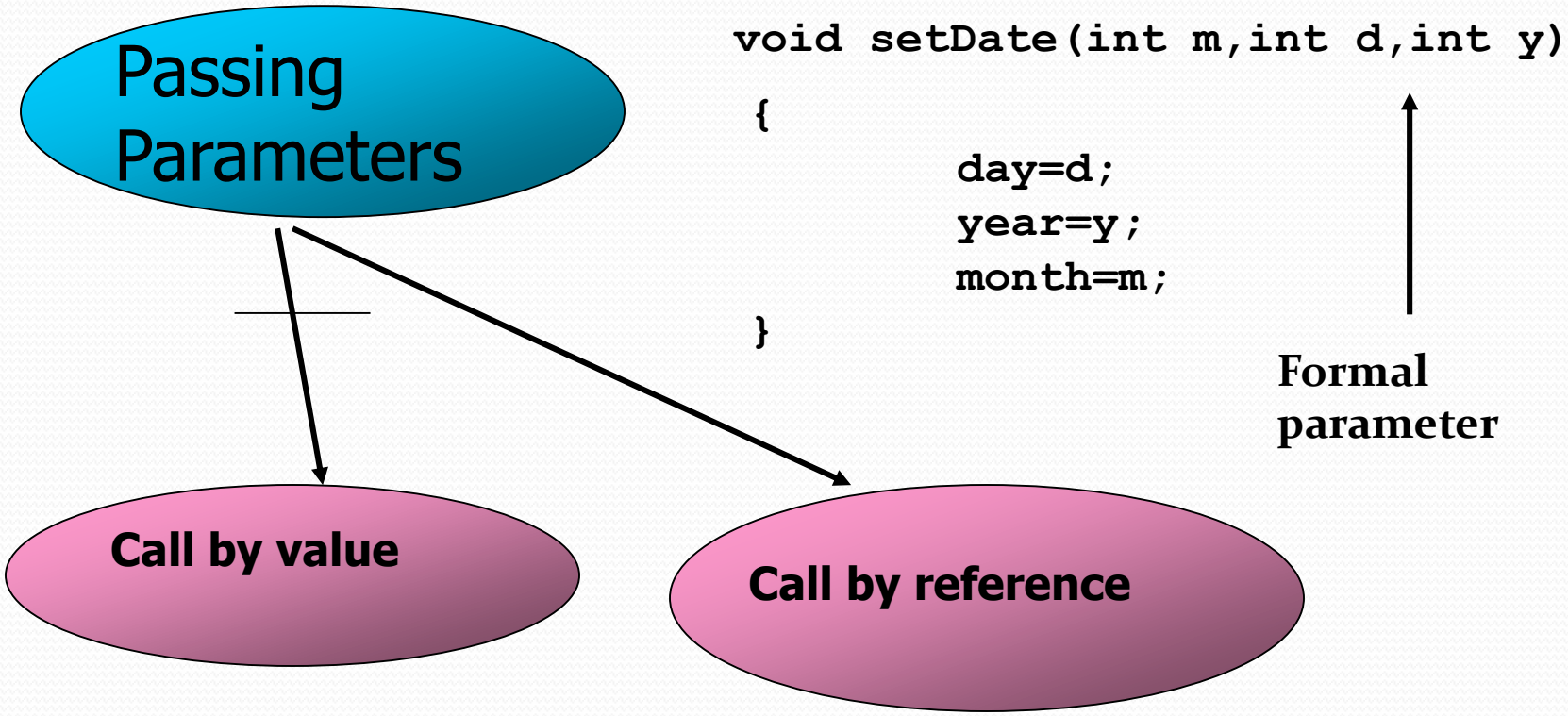
A method is always called to act on a specific object, not on the class in general

Example: **S1.setdate(27,1,1969)**

The general syntax for accessing a member function of a class is

Syntax: Class.object.method()

Passing Parameters



Call by Value

```
/* Example of Call by Value */
class Test {
    void change(int m,int n)
    {
     m = m * 2;
     n = n + 3;
      System.out.println("The  value  of  m  =  "+  m  +  "  and
      the value of n = "+n);
    }
  public static void main(String args[])
  {
    Test S1;
    S1=new Test();
    int a=10, b=12;
    System.out.println("Before : The value of a = " +a +" and the
    value of b = " + b);
    S1.change(a,b);
    System.out.println("After : The value of a = " +a + " and the
    value of b = " + b);
  }
 }
```

Call by Reference

```
class Test {
    int m,n;
    Test()      {
      m=10;
      n=20;
    }
    void change(Test T1)      {
      T1.m = T1.m * 2;
      T1.n = T1.n + 3;
    }
     public static void main(String args[])       {
      Test S1 = new Test();
      Test S2 = S1; //assigning reference of object S1 to S2
      System.out.println("Before : The value of m = "+S1.m + "and the
      value of n = " + S1.n);
      S1.change(S2);
      System.out.println("After : The value of m = "+ S1.m + " and the
      value of n = " + S1.n);
     }
}
```

Returning object from a method

A return statement in a function is considered to initialize a variable of the returned type

Syntax: **Test testobjectS1.func()**

```
test func ()
{
   test temp_object;
   .
   .
   return temp_object;
}
```

Accessor Functions

- Usually the data member are defined in private part of a class – information hiding
- Accessor functions are functions that are used to access these private data members
- Accessor functions also useful in reducing error

Example – Accessing Data Member

```
class Student{
  …
private int semester;
public void setRollNo(int aSem){
  if((aSem<1)||(aSem>8))
       System.out.println("Invalid Semester");
else
       semester = aSem;
  }
public int getCurrentSem()
{
  return semester;
}
}
```

Object-Oriented Programming

A ***class*** is a group of objects with the same properties and the same methods.

```
class Bicycle {

        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;
        }
```

Object-Oriented Programming

Each copy of an object from a particular class is called an ***instance*** of the object.

Object-Oriented Programming

The act of creating a new instance of an object is called **instantiation.**

Object-Oriented Programming

A class can be thought of as a blueprint for instances of an object.

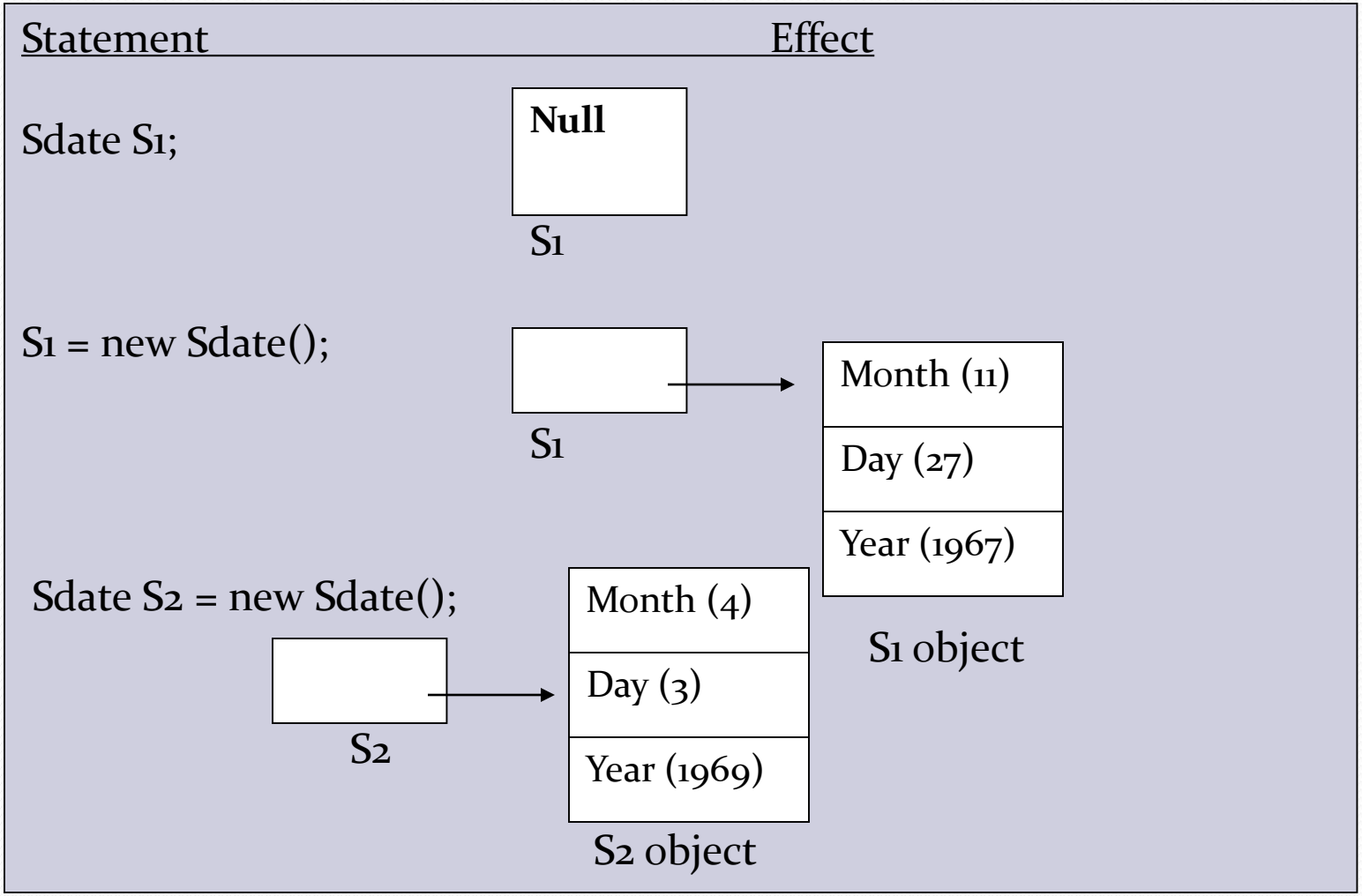
Object-Oriented Programming

Two different instances of the same class will have the same properties, but different values stored in those properties.

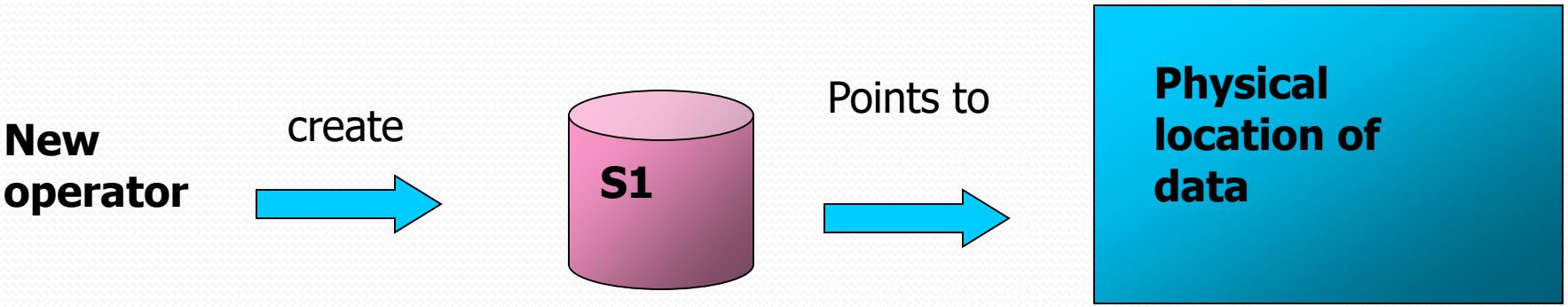
Using the Class

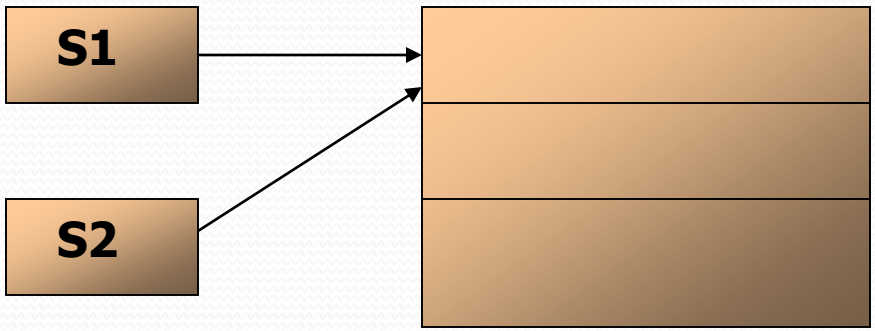
```
class Sdate {
                int month, day, year;
                void setDate(int m,int d,int y)
                {
                 month=m;
                 day=d;
                 year=y;
                }
           public static void main(String args[])
        {
        Sdate S1,S2;
        S1=new Sdate();
        S2=new Sdate();
        S1.setdate(11,27,1967);
        S2.setdate(4,3,1969);
          }
}
```

Defining Objects



Object Reference






```
class Sdate{
…….
……..
Sdate S1 = new Sdate();
Sdate S2 = S1;
……..
}
```

Class

- A Class defines an entity in terms of common characteristics and actions

Class Customer
Name of the customer
Address of the customer
Model of the car bought
Salesman's name who sold the car
Accept Name
Accept Address
Accept Model of the car purchased
Accept the name of the salesman who sold the car
Generate the bill

Messages

- Objects communicate through messages
- They send messages (stimuli) by invoking appropriate operations on the target object
- The number and kind of messages that can be sent to an object depends upon its interface

Examples – Messages

- A Person sends message (stimulus) “stop” to a Car by applying brakes
- A Person sends message “place call” to a Phone by pressing appropriate button

Object

- Attribute
  - Characteristic that describes an object
- Operation
  - Service that can be requested of an object
- Method
  - Specification of how the requested operation is carried out
- Message
  - Request for an operation
- Event
  - Stimulus sent from one object to another

Class vs. Object

- Class defines an entity, while an object is the actual entity
- Class is a conceptual model that defines all the characteristics and actions required of an object, while an object is a real model
- Class is a prototype of an object
- All objects belonging to the same class have the same characteristics and actions

Class and Objects – Example

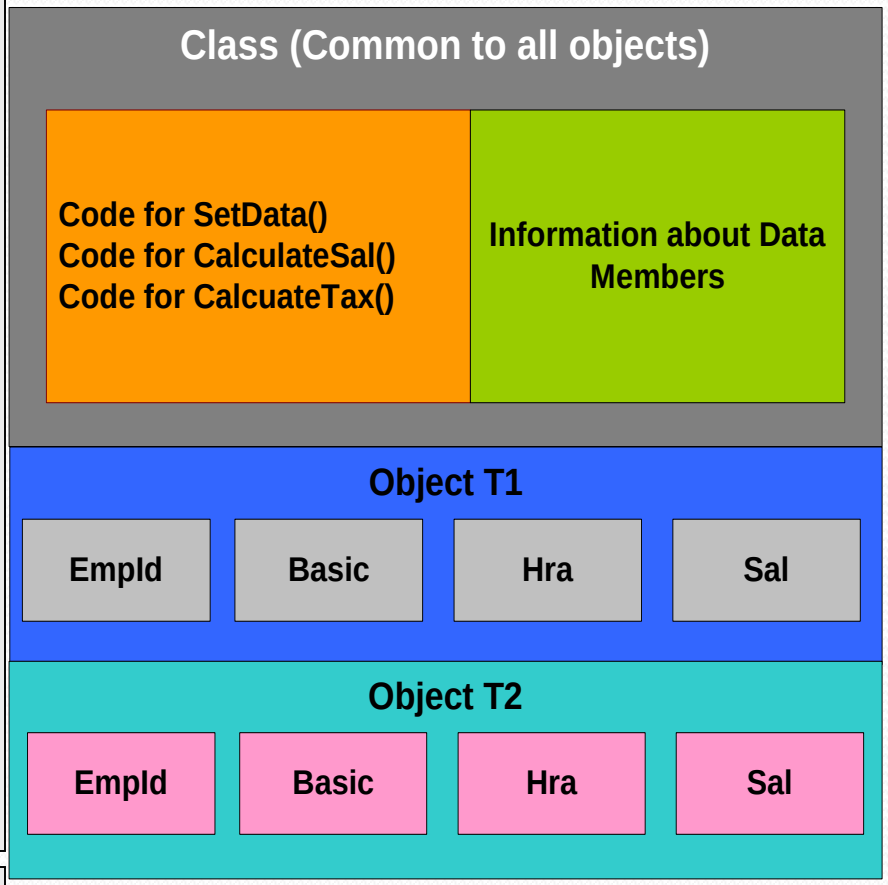
```
class Trainee {
private   int empId;
private    String empName;
private   float basic;
private   float hra;
public void SetData(int iEmpId,
        String acEmpName, float fBasic,
        float  fHRA) {
              .....       }
public  void CalculateSal() {
              .....       }
public     void CalculateTax() {
              .....}
} //class Trainee ends here
```

```
public static void main(String [] args) {
     /* Object Creation */
     Trainee oT1 = new Trainee();

     /* Invoking SetData */
     oT1.SetData(101,"Hamza",1200,150)

     /* Invoking CalculateSal */
     oT1.CalculateSal();

     /* Invoking CalculateTax */
     oT1.CalculateTax();

}
```

Memory allocation for Classes and Objects

```
class Trainee {
private     int m_iEmpId;
private     float m_fBasic;
private     float m_fHRA;
private     float m_fSalary;
public void SetData(int iEmpId, float fBasic,
            float  fHRA){
      }
public  void CalculateSal() {
              // code goes here
     }
public  void CalculateTax() {
              //code goes here
     }
}
```

```
Trainee oT1 = new Trainee();
Trainee oT2= new Trainee();
```



Object-Oriented Programming

Message

Object

Property

The same terminology is used in most object-oriented programming languages.

Instantiation

Method

Class

Instance

State