# Introducing Objects and Object Oriented Programming

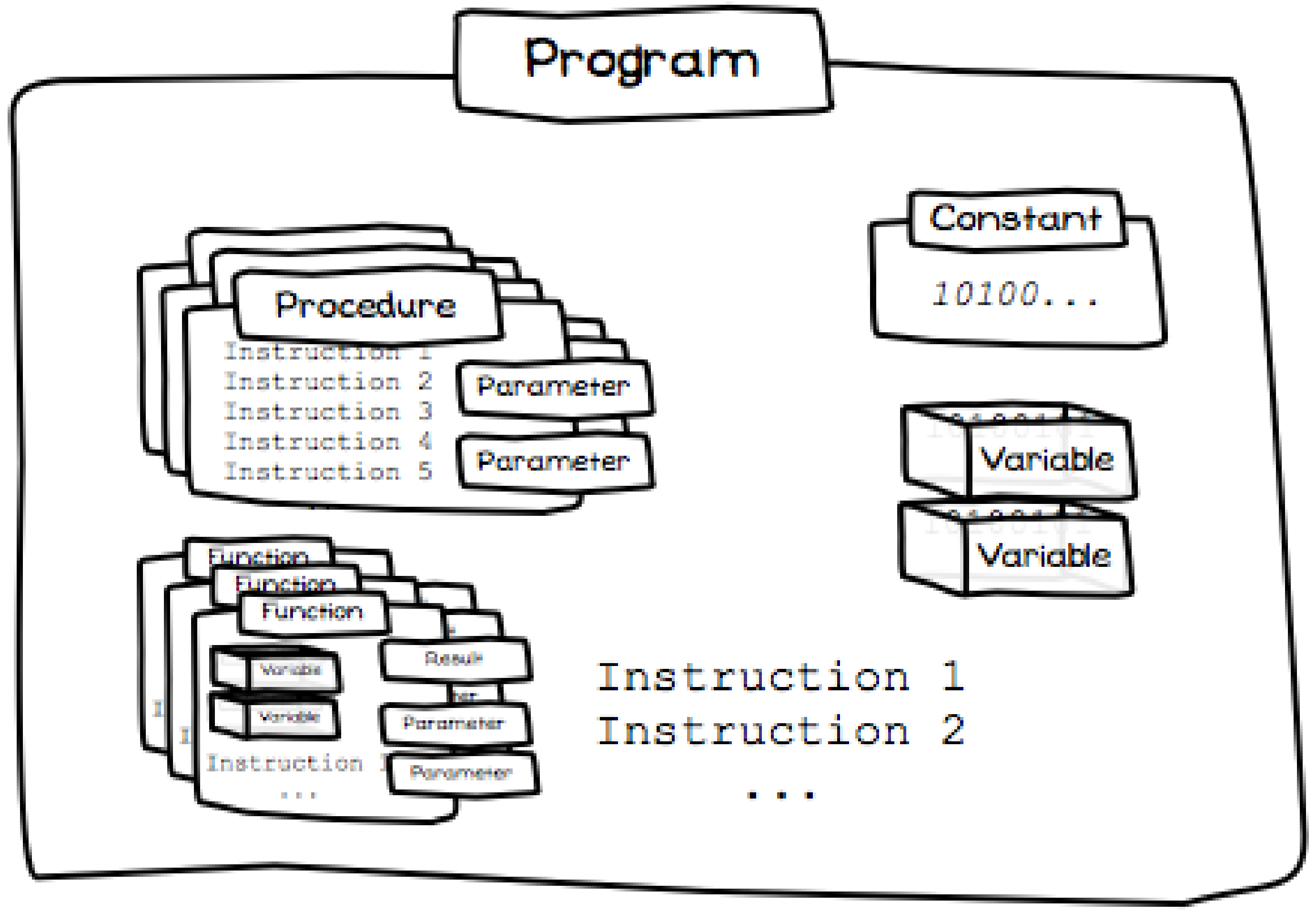
by Andrew Cain and Willem van Straten

Object Oriented Programming
Object Oriented Programming in C++

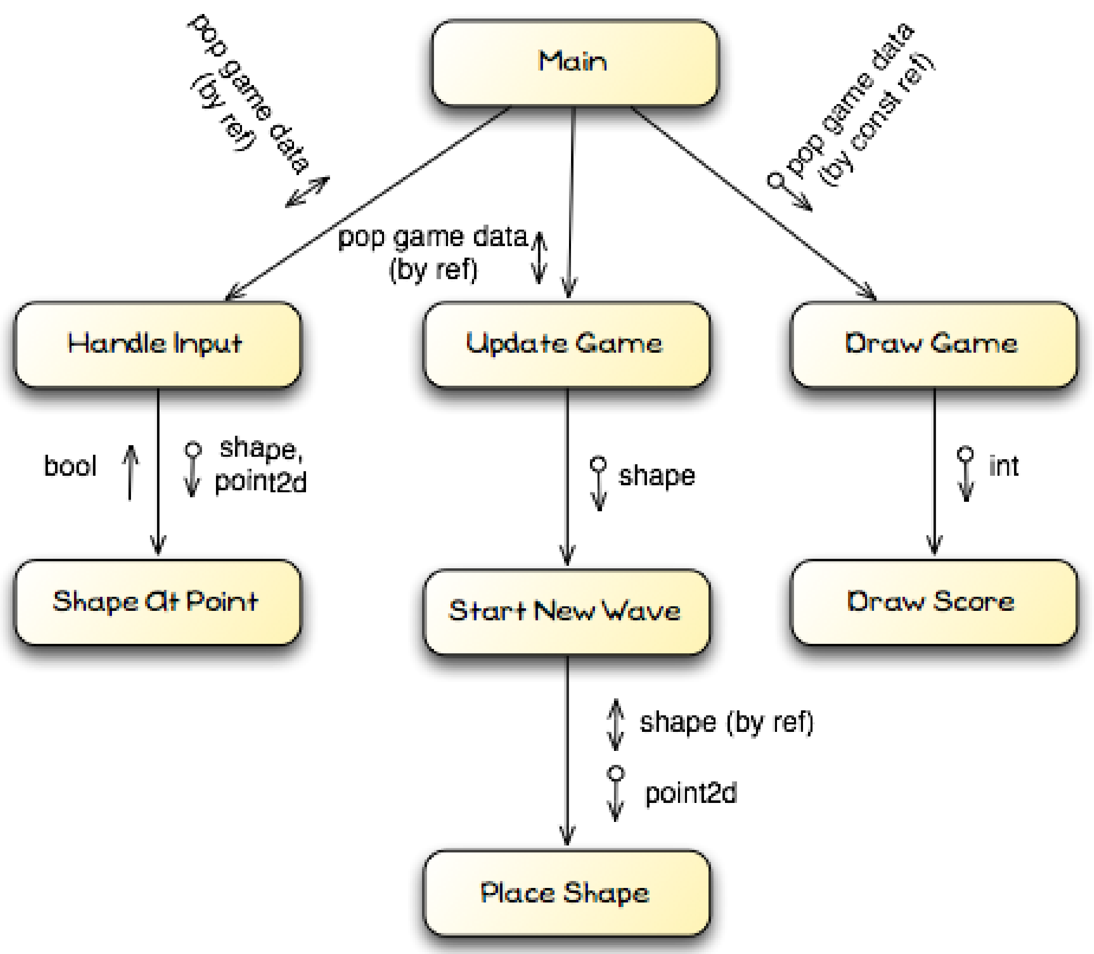
# Software Development is about defining instructions for computers

Code $\xrightarrow{\text{Instructs}}$

# Developers create programs using a range of artefacts to manage complexity

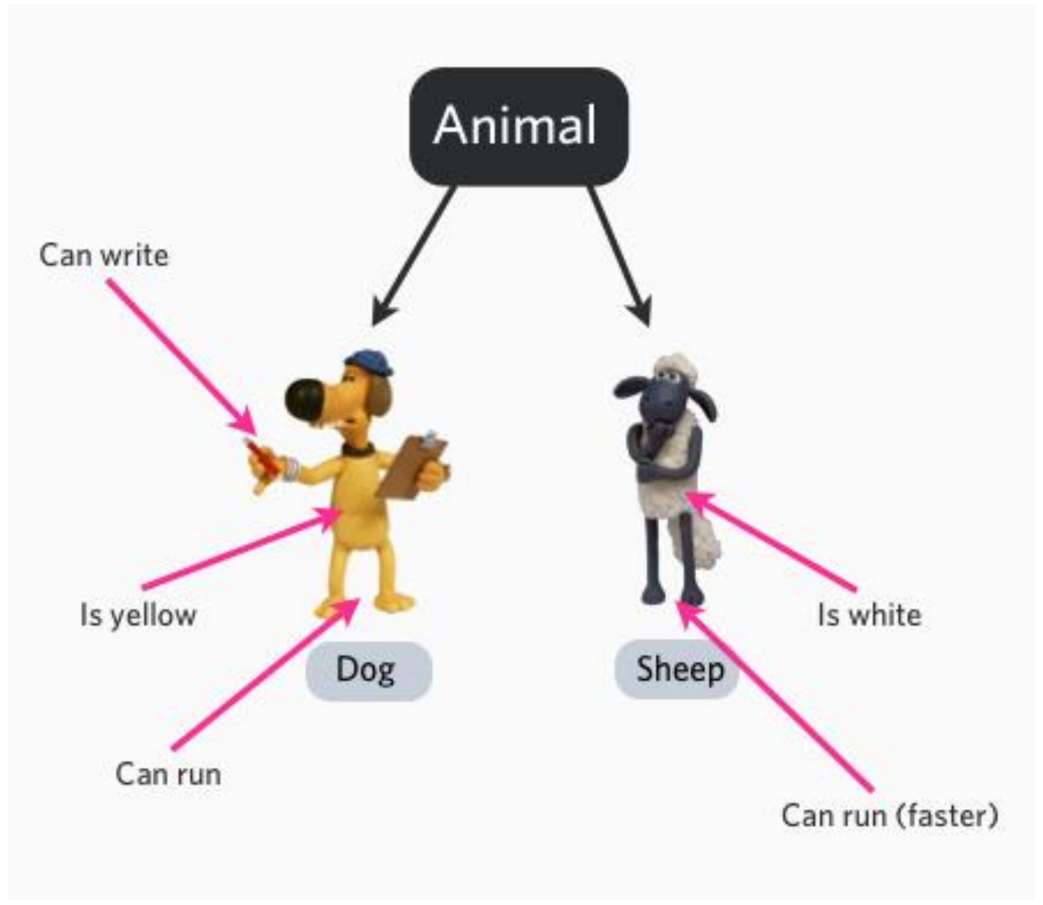


# Procedural programming uses functional decomposition, but has limits as size grows



## Object oriented programming offers means of managing complexity for larger software

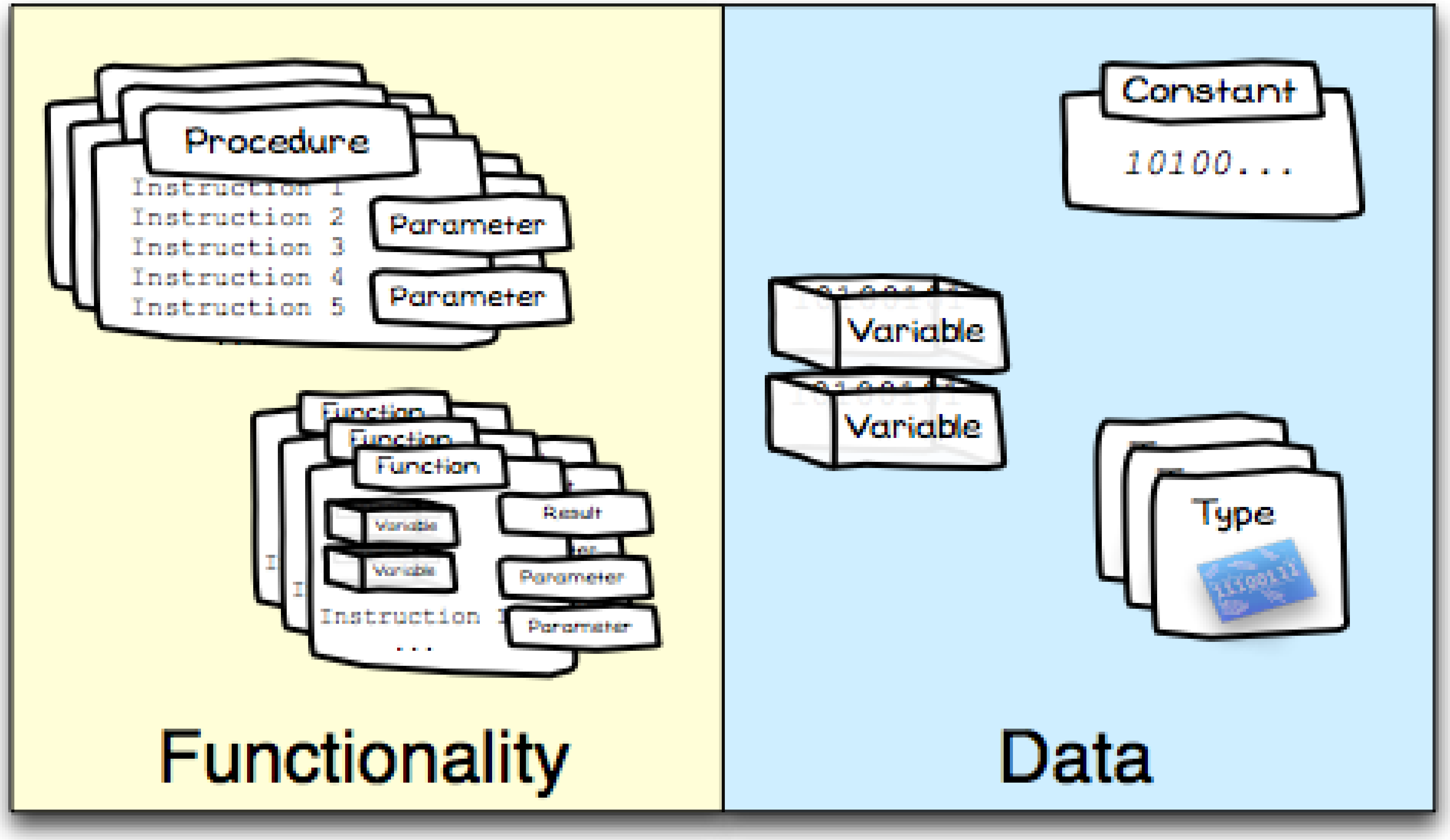
- Objects are the central idea behind OOP !!
- breaks problems down into small parts
- objects are larger than single functions
- capture both data and functionality
  - contains variables and related methods
- easier to manage larger software systems
- reusable



## Start to master Object Oriented Programming by seeing that objects **know** and **do** things

See the difference between the Object Oriented Paradigm and the Procedural Paradigm

# Procedural Programming focuses on data and functionality **separately**



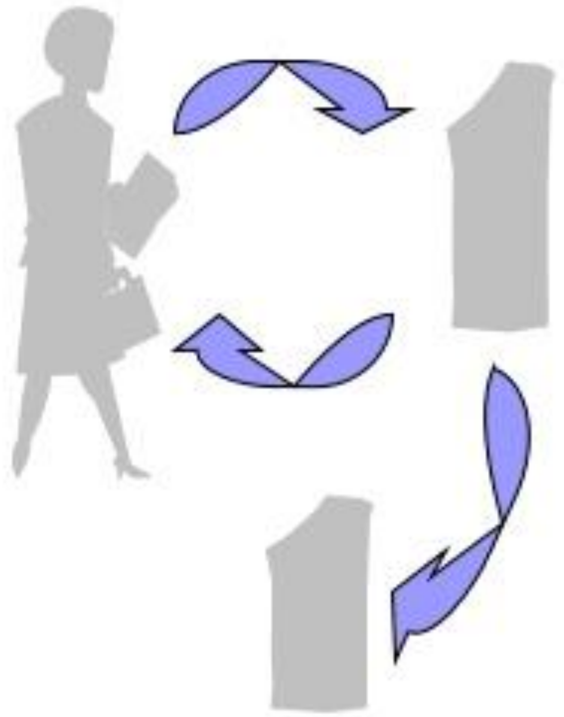
Not reusable

**Harder to maintain**

# Example

## Procedural vs. Object-Oriented

- Procedural

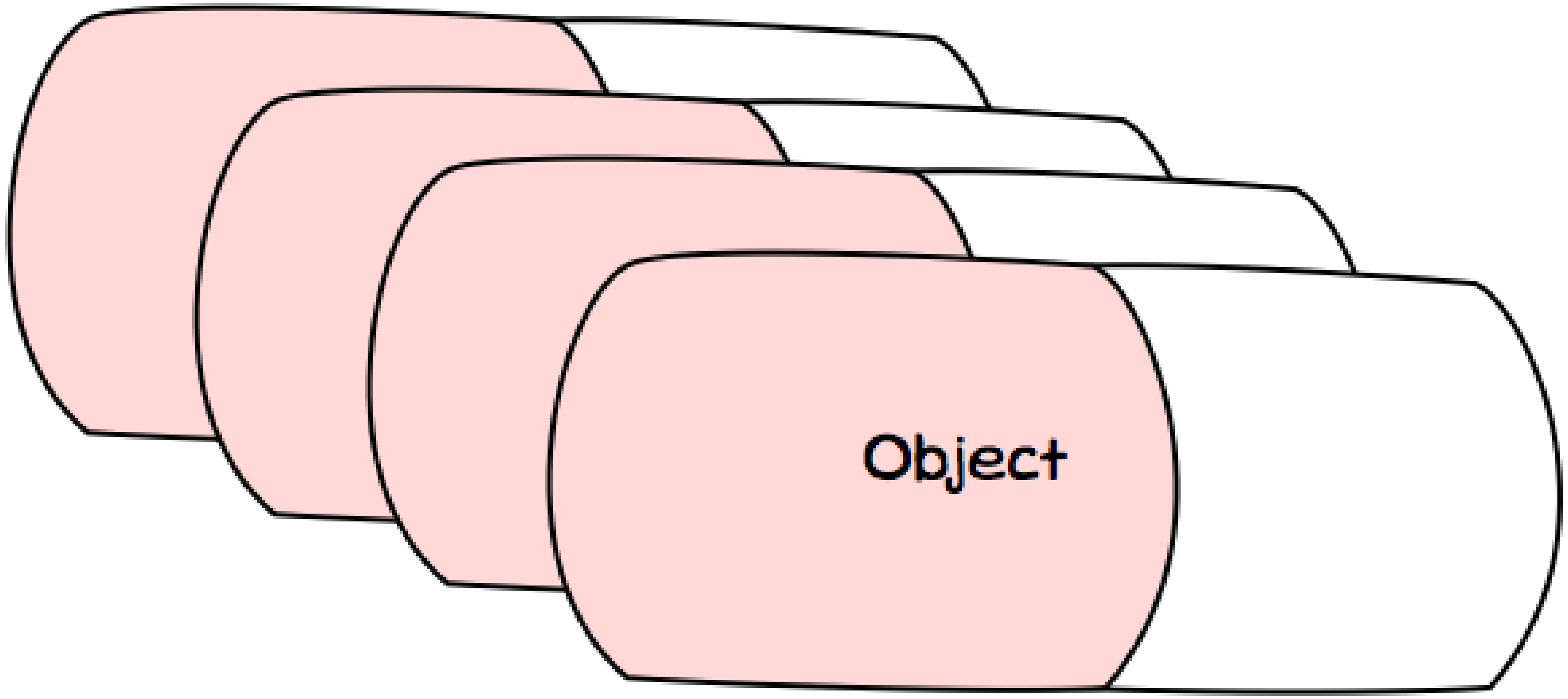


Withdraw, deposit, transfer

- Object Oriented

Customer, money, account

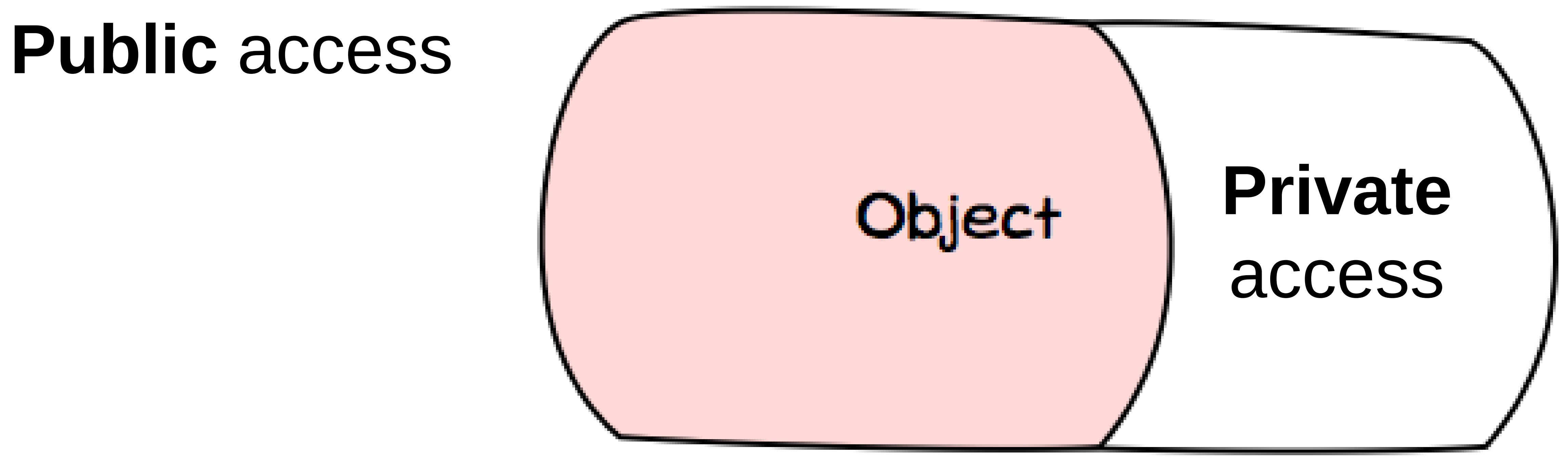
# Object oriented programming introduces higher level artefacts: Objects



## Objects **encapsulate** both functionality and data - they *know* and can *do* things

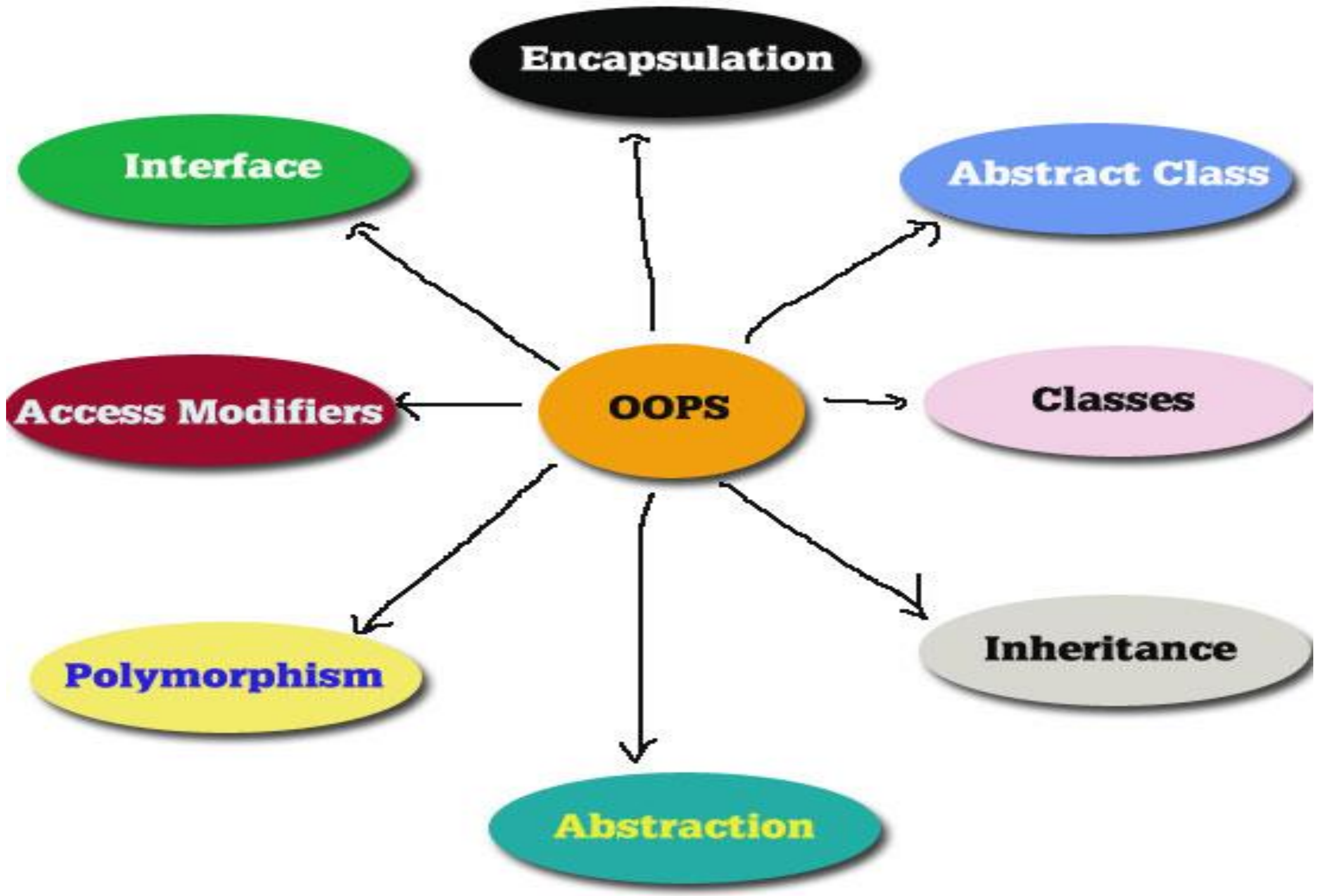
Object
Data
&
Functionality
Know things
&
Do things

# Remember capsules have an "inside" and "outside" - not everything is accessible



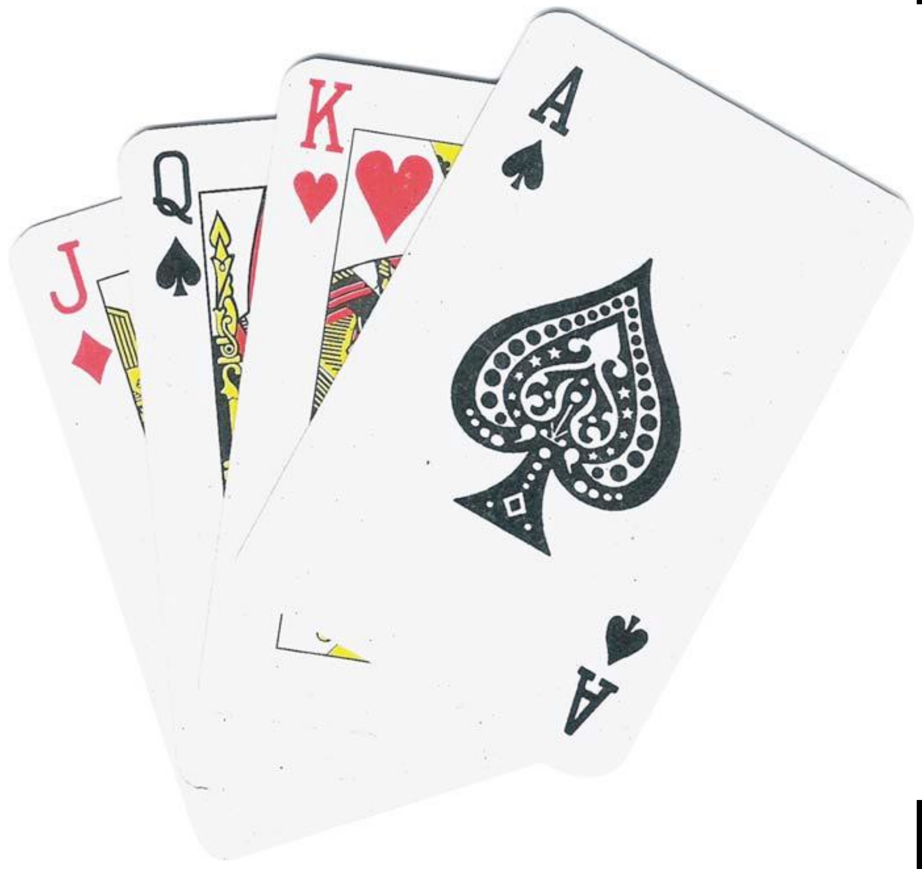
Things the object *knows* and *can do* can be **hidden** within the object.

# Object-oriented concepts



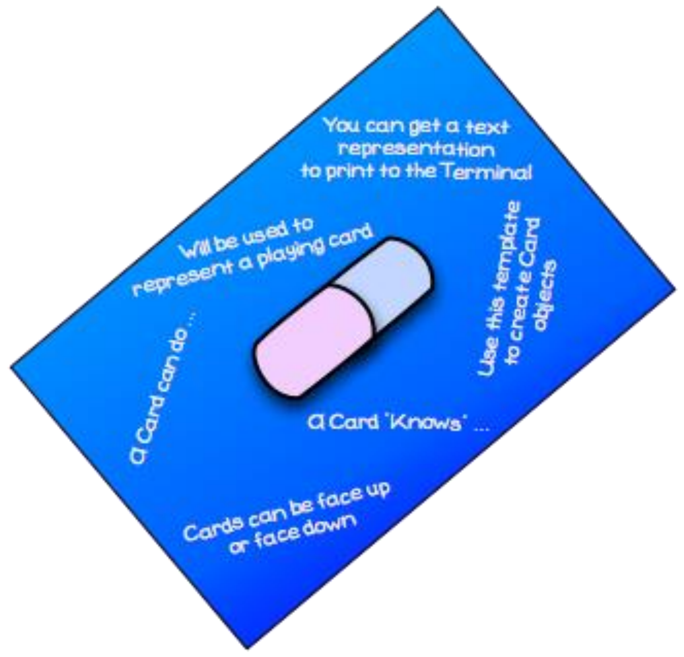
Design programs by breaking problems down into objects

# Use **abstraction** to classify objects for your program



Classification

Specification for a Card

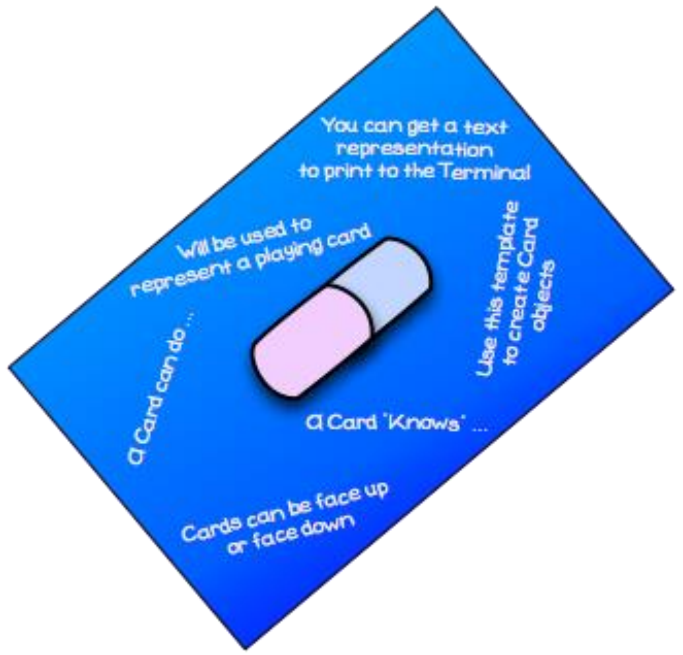

You can get a text representation to print to the Terminal
Will be used to represent a playing card
Use this template to create Card objects
a Card can do ...
a Card 'Knows' ...
Cards can be face up or face down

**Abstraction** is the process of identifying the essential aspect of an entity and ignoring the unimportant properties.

# Use these classifications to **create objects** at run time

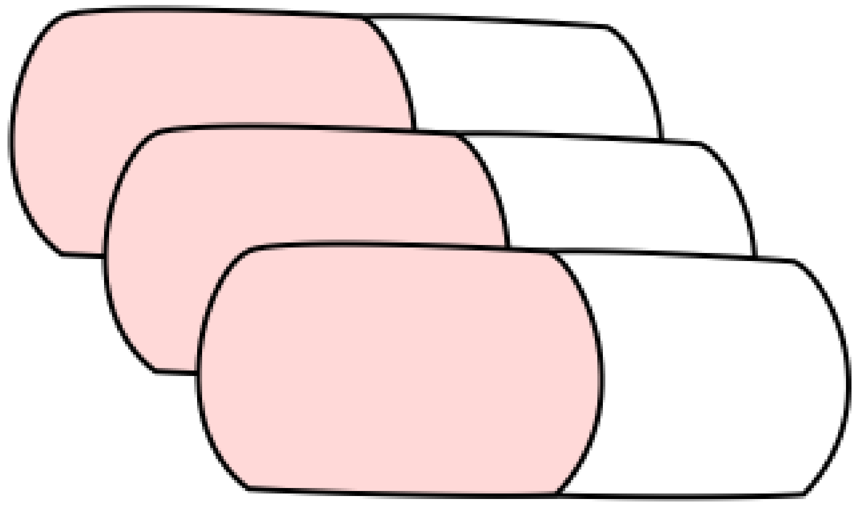
Classification

Specification for a Card

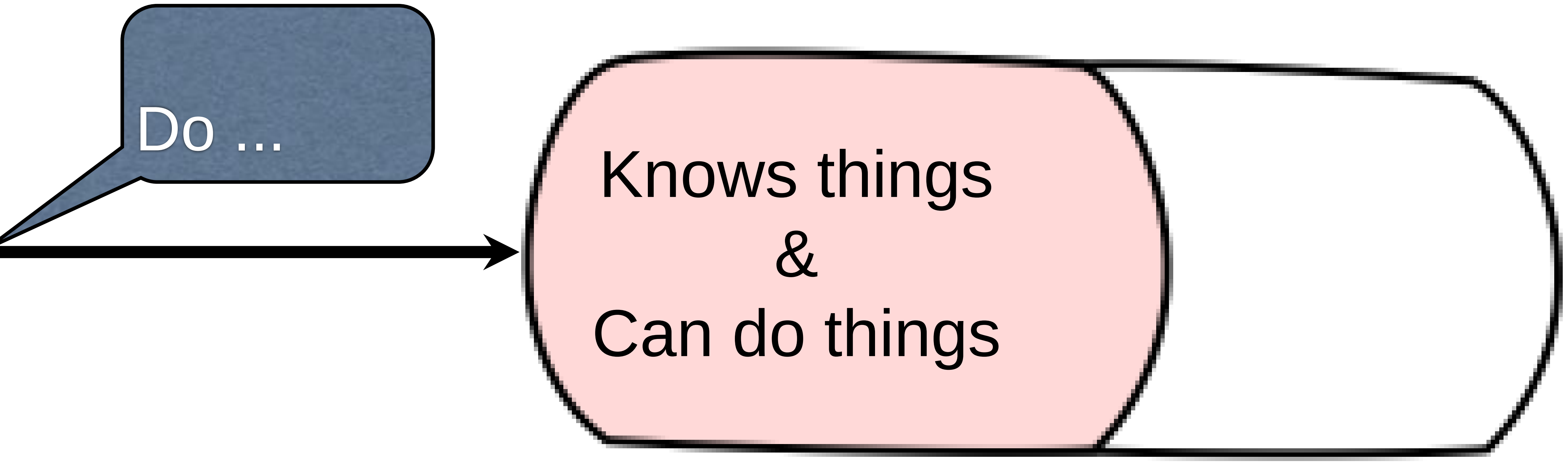

You can get a text representation to print to the Terminal
Will be used to represent a playing card
a Card can do ...
Use this template to create Card objects
a Card 'Knows' ...
Cards can be face up or face down

Creation

Card Objects

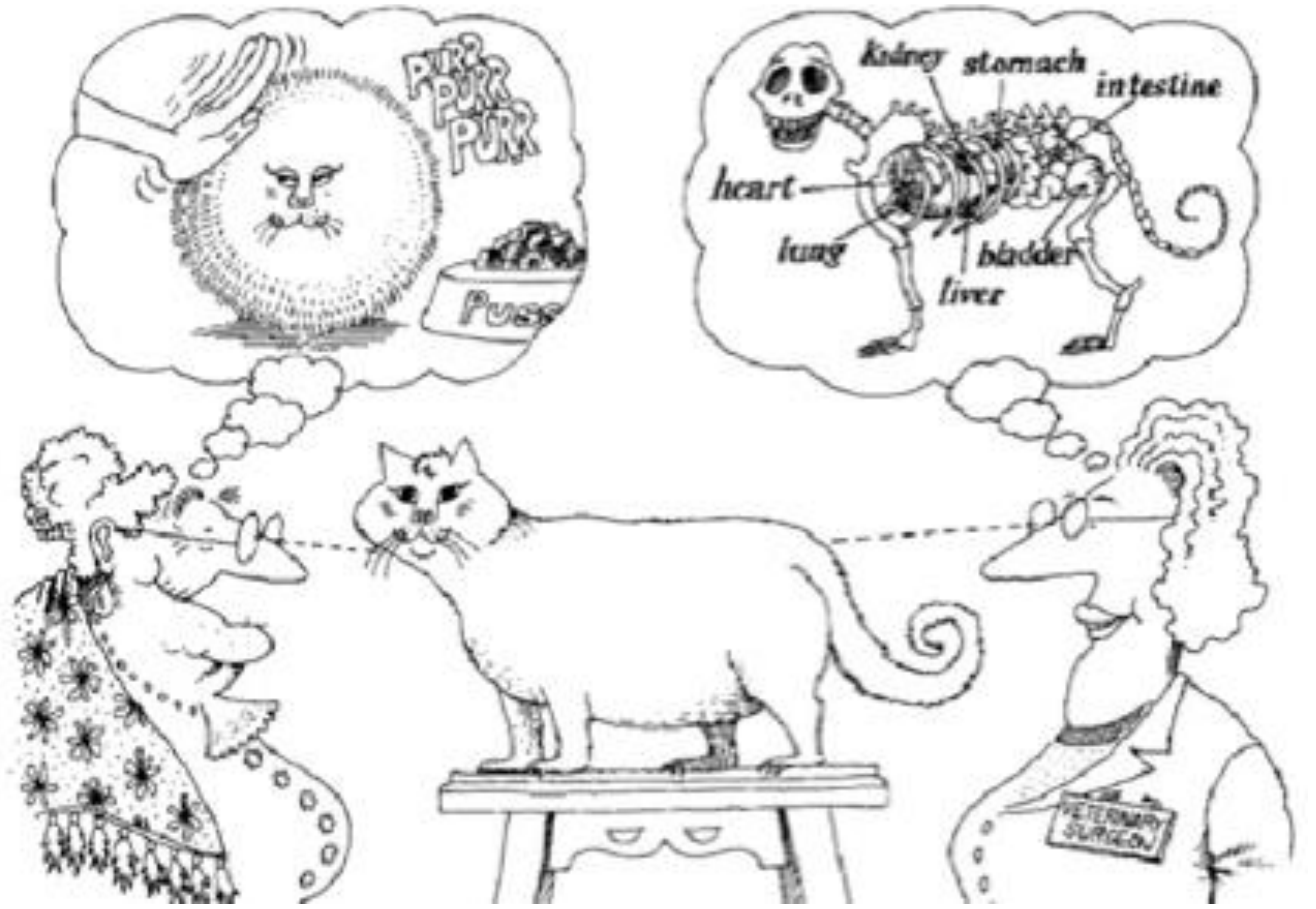


# Tell objects to *do things to* accomplish program goals



Uses what it knows
to achieve the
requested task

# A little bit more about Abstraction…



## Abstraction:

- is outlined by the top left and top right images of the cat.
- The surgeon and the old lady designed (or visualized) the animal differently.
- In the same way, you would put different features in the Cat class, depending upon the need of the application.
- Every cat has a liver, bladder, heart and lung, but if you need your cat to 'purr' only, you will abstract your application's cat to the design on top-left rather than the top-right.

*Retrieved from web source: http://stackoverflow.com/questions/16938667/how-abstraction-and-encapsulation-differ*

## Basic object features

- *Attribute – characteristics of an object*
- *Behaviour – action that an object is capable of performing*

# Class Activity 1

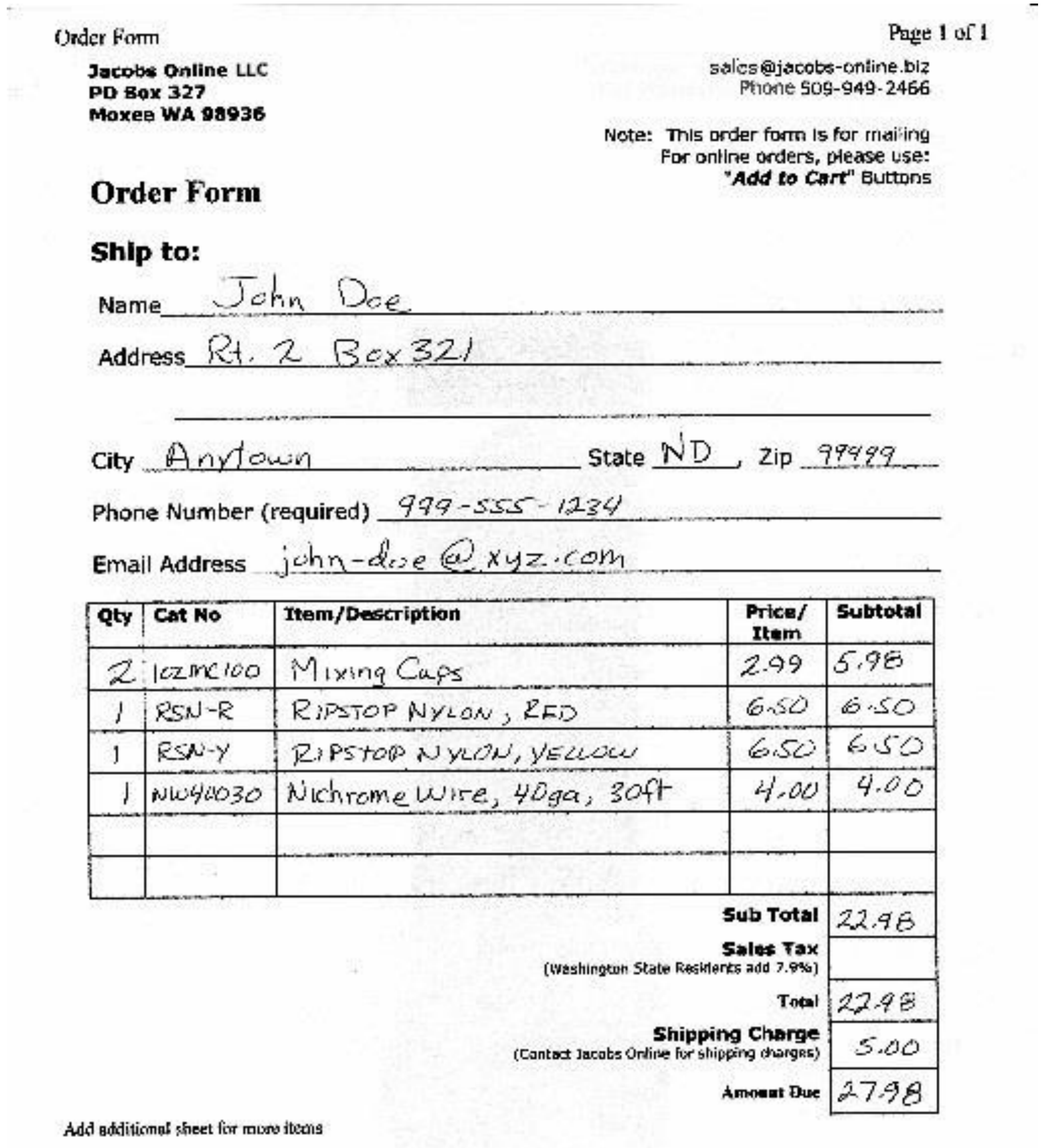
Suppose you wish to write a computer soccer games. It is quite difficult to model the game in procedural-oriented languages. But using OOP languages, you can easily model the program accordingly to the "real things" that appear in the soccer games. In this case,

| Name |
|---|
| Attributes |
| Behaviour |

- identify the objects, their attributes and behaviour.
- identify the classes that can be reused in another application eg. baseball

## Class Activity 2

From the order form shown given, identify the objects and their attributes.


Order Form

Page 1 of 1

**Jacobs Online** LLC
**PO Box 327**
**Moxee WA 98936**

sales@jacobs-online.biz
Phone 509-949-2466

Note: This order form is for mailing
For online orders, please use:
"***Add to Cart***" Buttons

**Order Form**

**Ship to:**

Name John Doe

Address Rt. 2 Box 321

City Anytown State ND, Zip 99999

Phone Number (required) 999-555-1234

Email Address john-doe@xyz.com

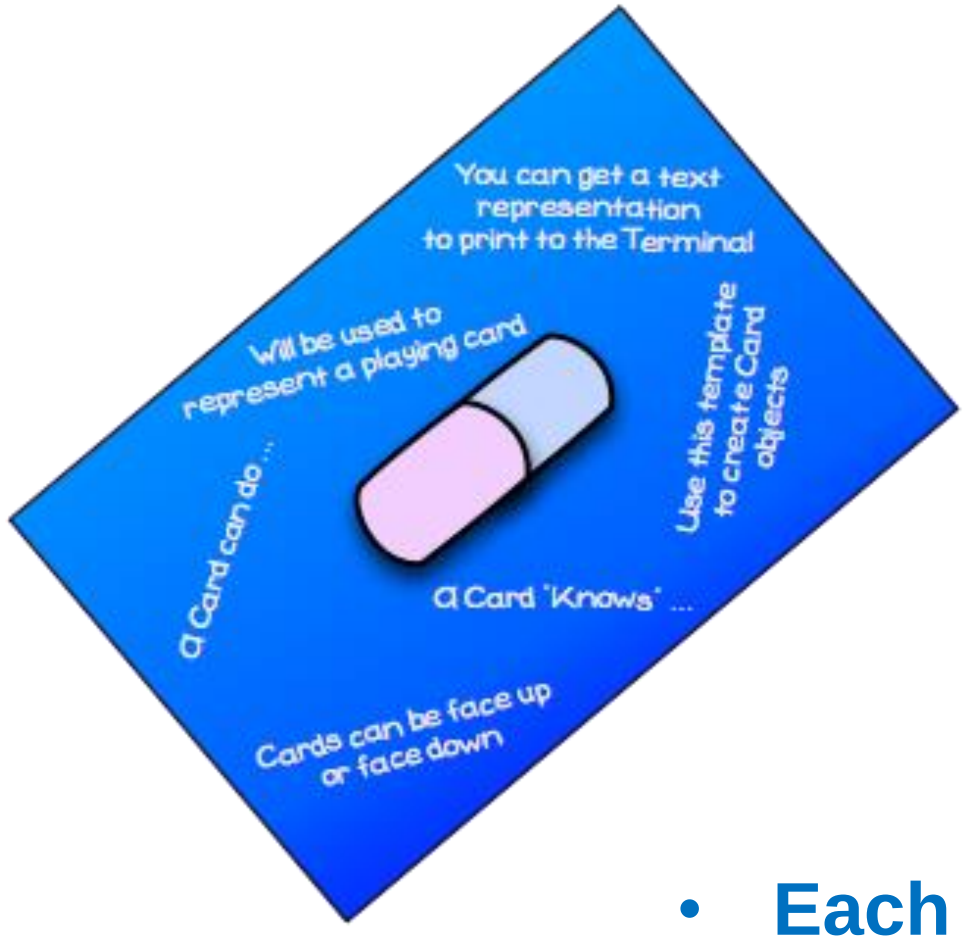
| Qty | Cat No | Item/Description | Price/ Item | Subtotal |
|---|---|---|---|---|
| 2 | 1czmc100 | Mixing Cups | 2.99 | 5.98 |
| 1 | RSN-R | RIPSTOP NYLON, RED | 6.50 | 6.50 |
| 1 | RSN-Y | RIPSTOP NYLON, YELLOW | 6.50 | 6.50 |
| 1 | NW40030 | Nichrome Wire, 40ga, 30ft | 4.00 | 4.00 |
| | | | | |
| | | | | |

| | |
|---|---|
| **Sub Total** | 22.98 |
| **Sales Tax** (Washington State Residents add 7.9%) | |
| **Total** | 22.98 |
| **Shipping Charge** (Contact Jacobs Online for shipping charges) | 5.00 |
| **Amount Due** | 27.98 |

Add additional sheet for more items

Implement your designs using an object oriented programming language

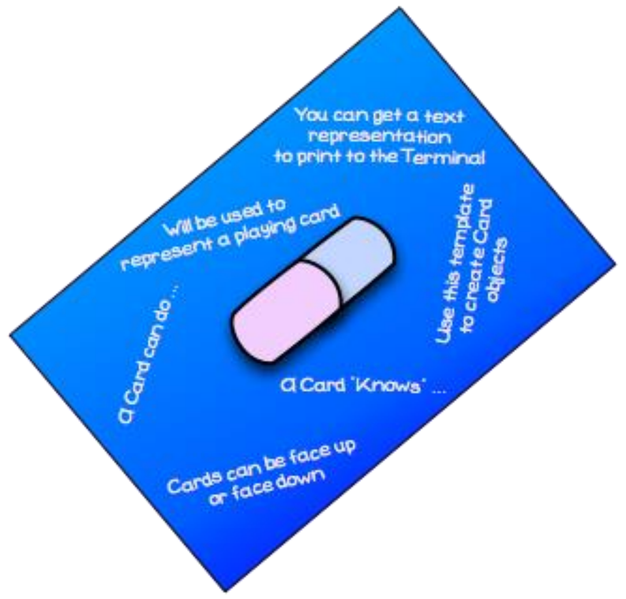
# In each case, define **classes** to capture object specifications

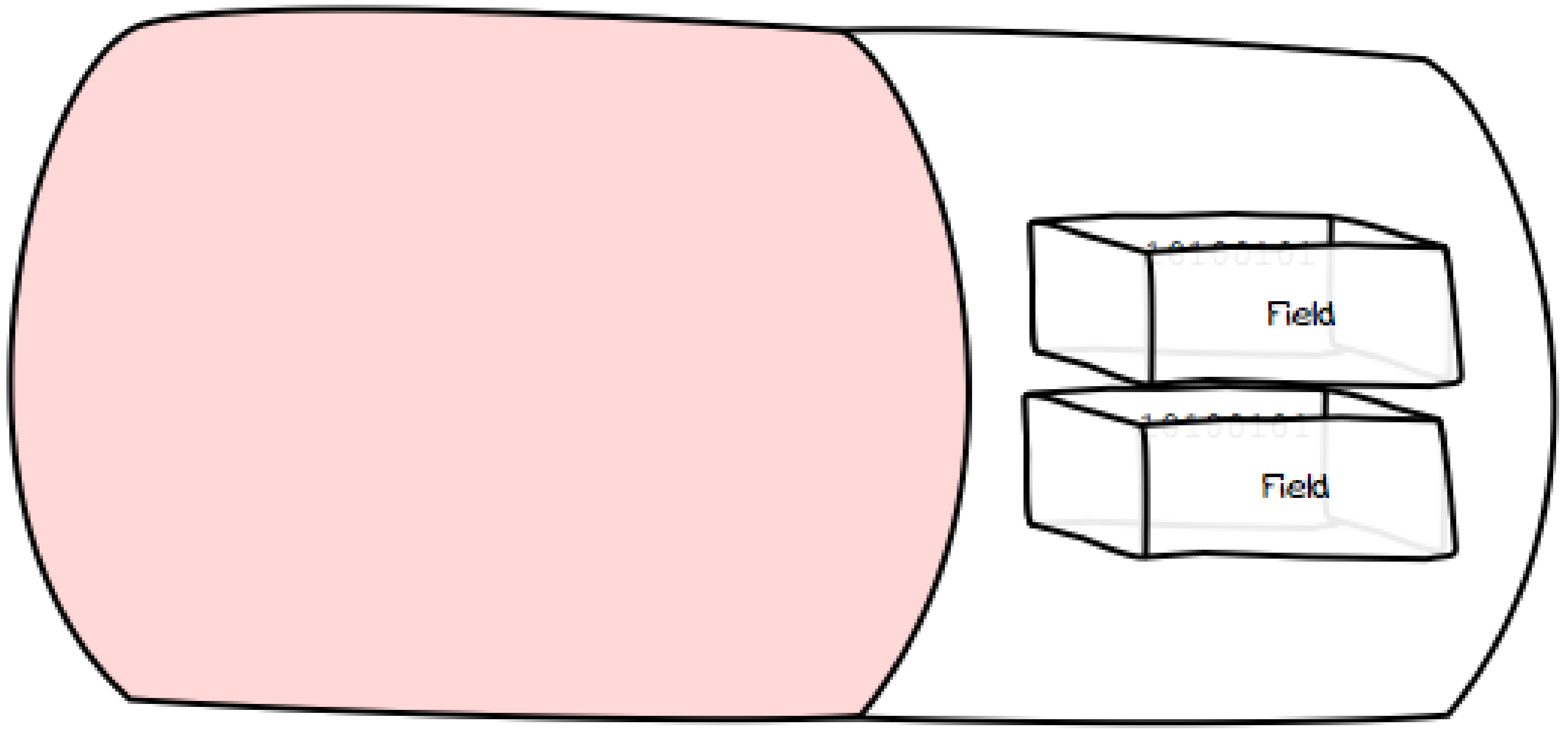

You can get a text representation to print to the Terminal
Will be used to represent a playing card
Use this template to create Card objects
a Card can do ...
a Card 'Knows' ...
Cards can be face up or face down

`class`

- **Each class is like a template for object creation.**
- **You write code for the class, then use this to create objects at runtime.**

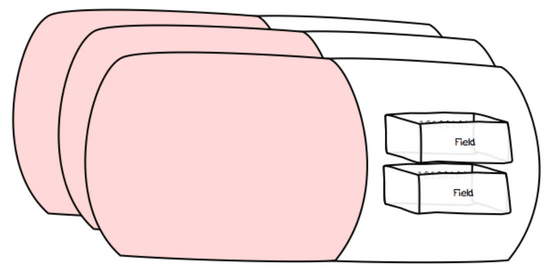
## Add private **fields** to the class to store the things the object *knows*


You can get a text representation to print to the Terminal
Will be used to represent a playing card
Use this template to create Card objects
a Card can do ...
a Card 'Knows' ...
Cards can be face up or face down

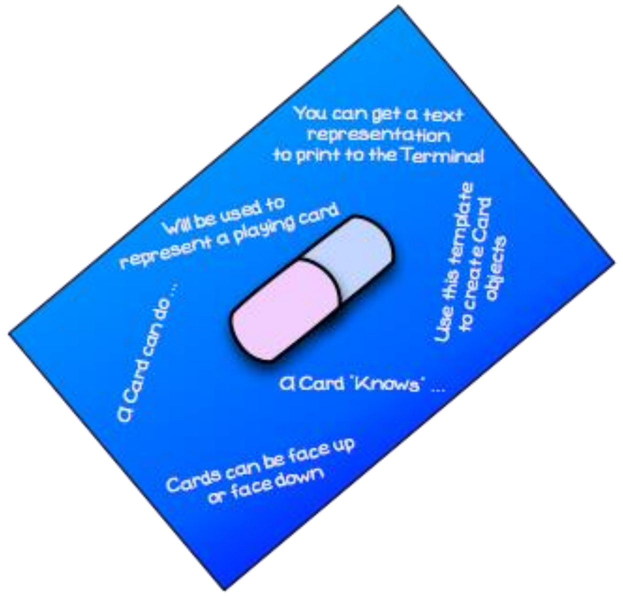


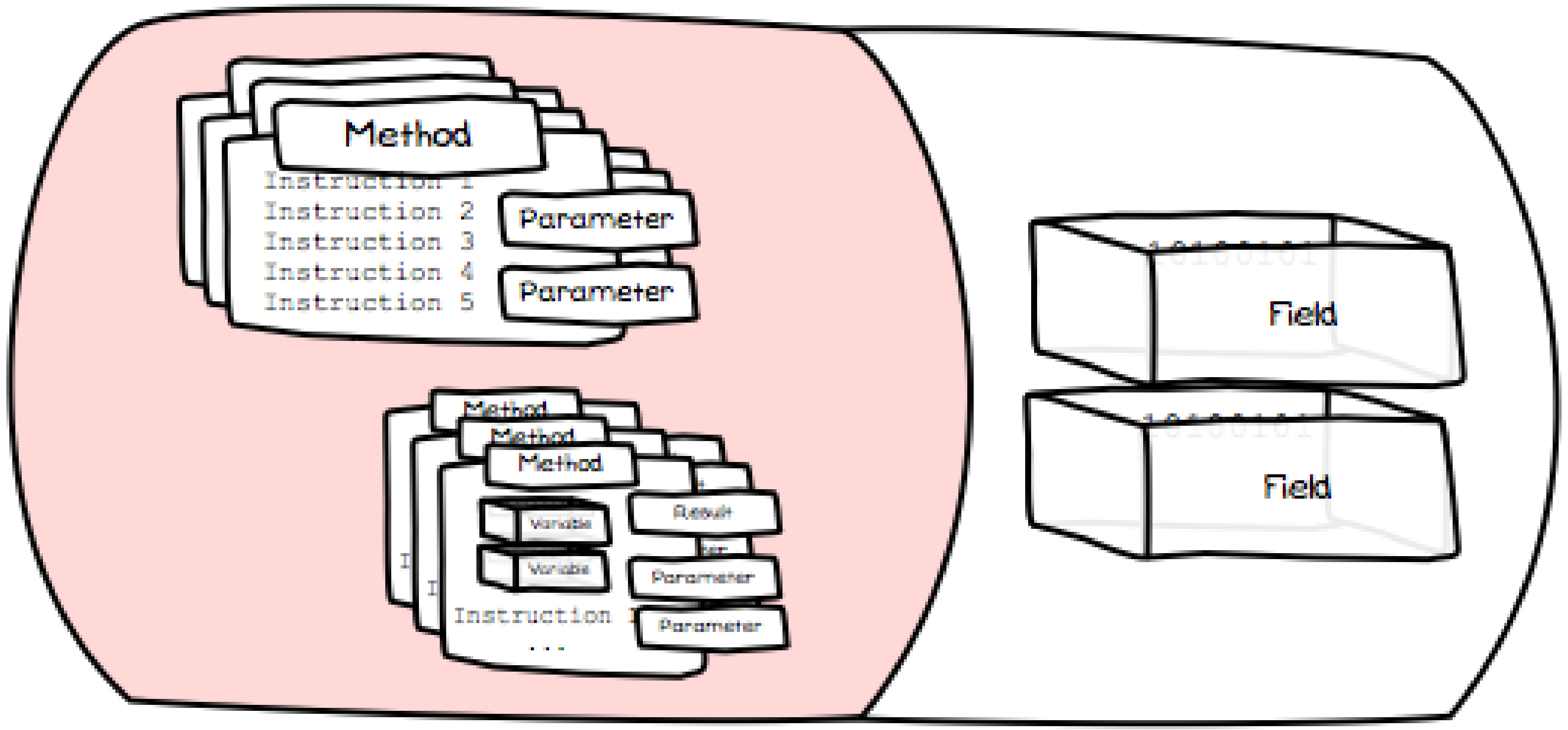
Fields (aka attributes/variables) are declared within the class

The field exists within each object



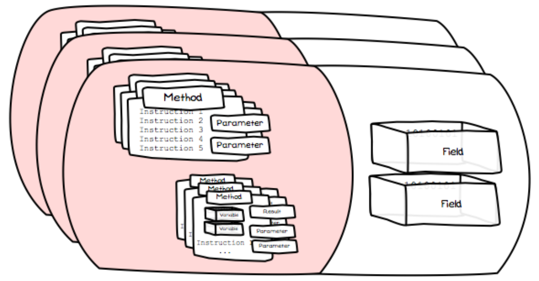
# Add **methods** to the class to code the things the object *can do*


You can get a text representation to print to the Terminal
Will be used to represent a playing card
Use this template to create Card objects
A Card can do ...
A Card 'Knows' ...
Cards can be face up or face down



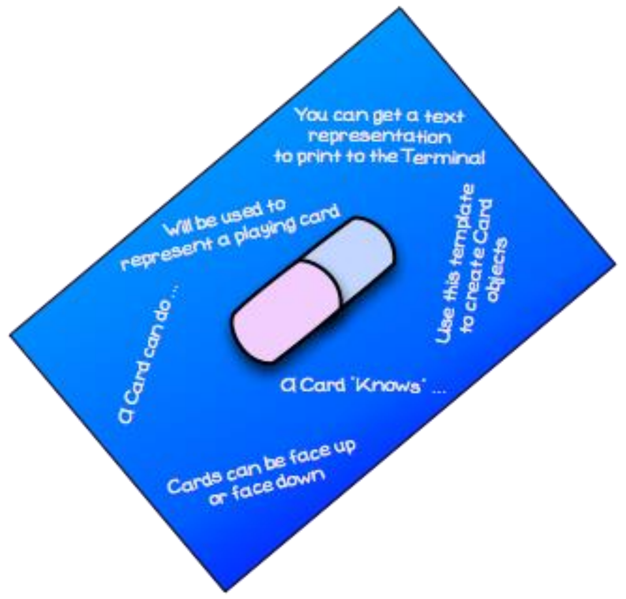
Methods are declared within the class

The methods exist for each object

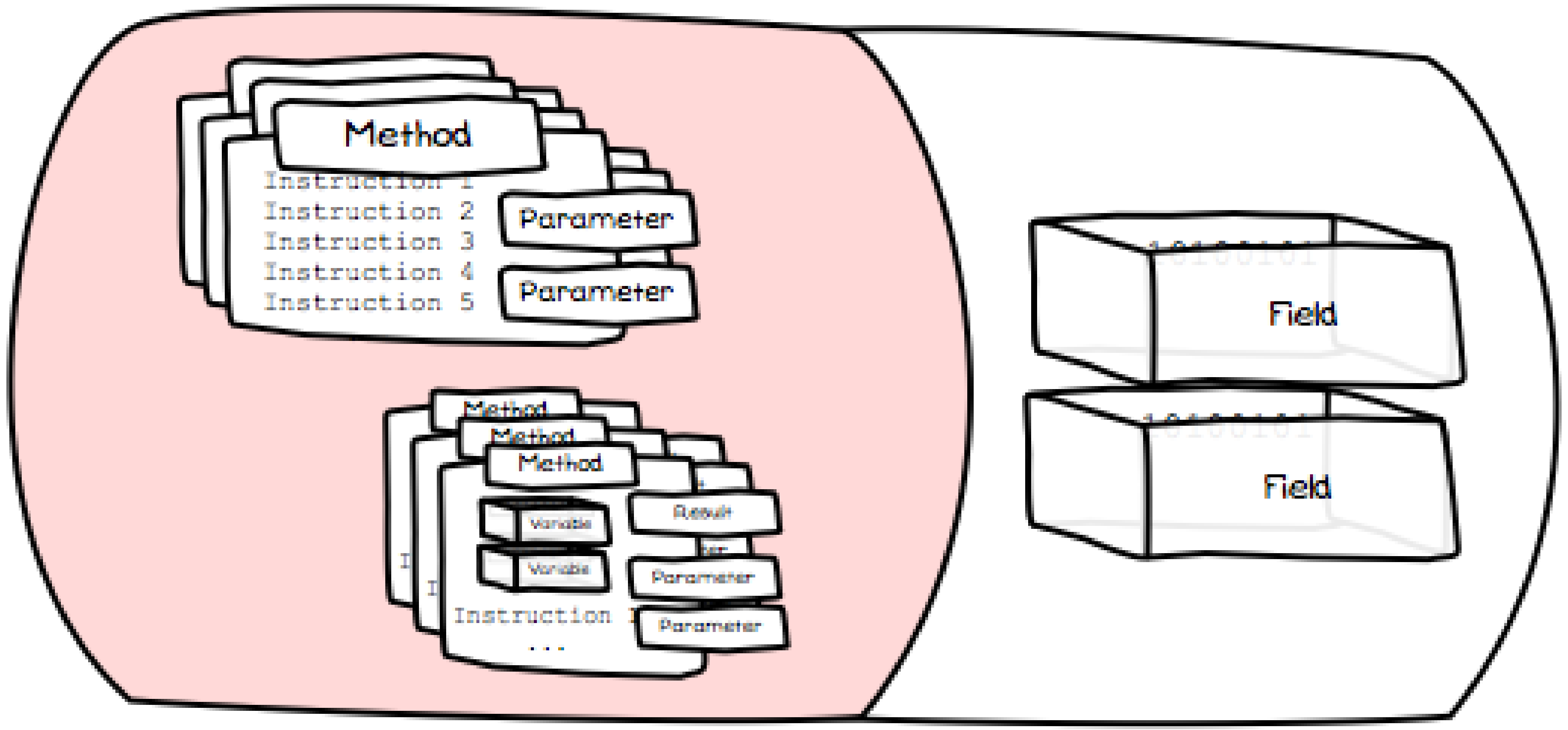


## Add **properties** to the class to provide access to hidden data

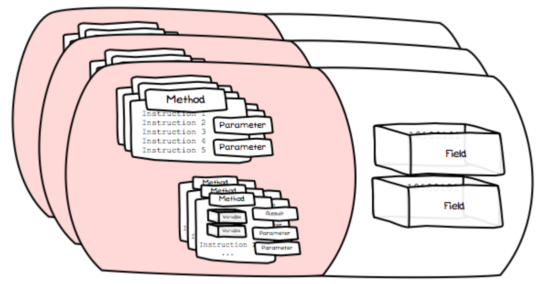
Also termed as **encapsulation**


You can get a text representation to print to the Terminal
Will be used to represent a playing card
Use this template to create Card objects
a Card can do ...
a Card 'Knows' ...
Cards can be face up or face down

Properties are **get** and **set** methods (***accessor methods***) declared within the class



The properties exist for each object

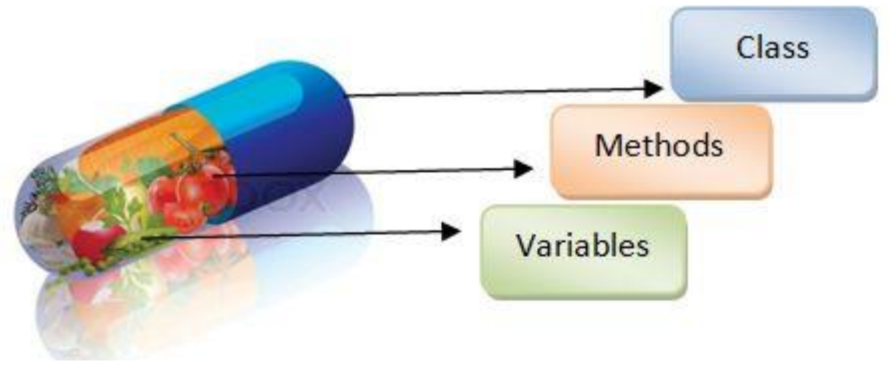


## More on “properties”- Access Specifier

Access Specifiers (Access Modifiers) describes the scope of accessibility of an Object and its members

- **public**: It can be accessed from anywhere (inside and outside class), that means there is no restriction on accessibility.
- **protected**: accessibility is limited within the class or *struct* and the class derived (inherited )from this class.
- **private**: Accessibility is strictly limited to only inside the classes.

## By using encapsulation,



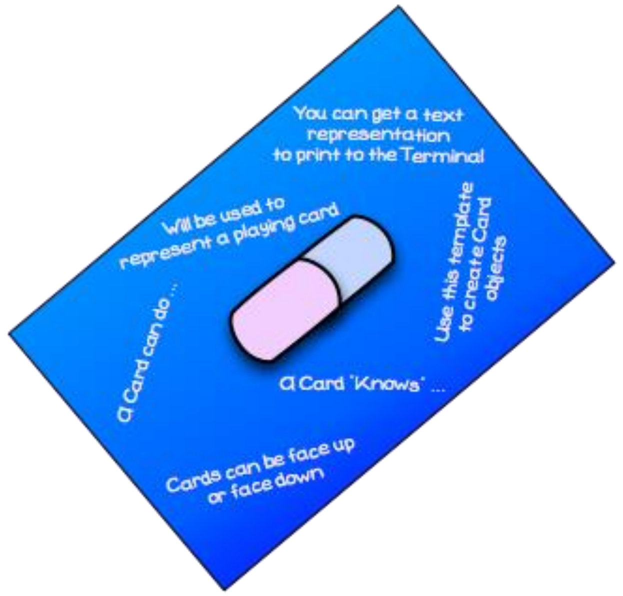
- Object can expose only the data and methods necessary to other objects, while hiding its irrelevant fields and methods.

- Achieved through:
  ✓ "private" access specifier
  ✓ the use of accessors (to get data) and mutators (to modify data) (a.k.a "properties" in C#)

## Example of properties

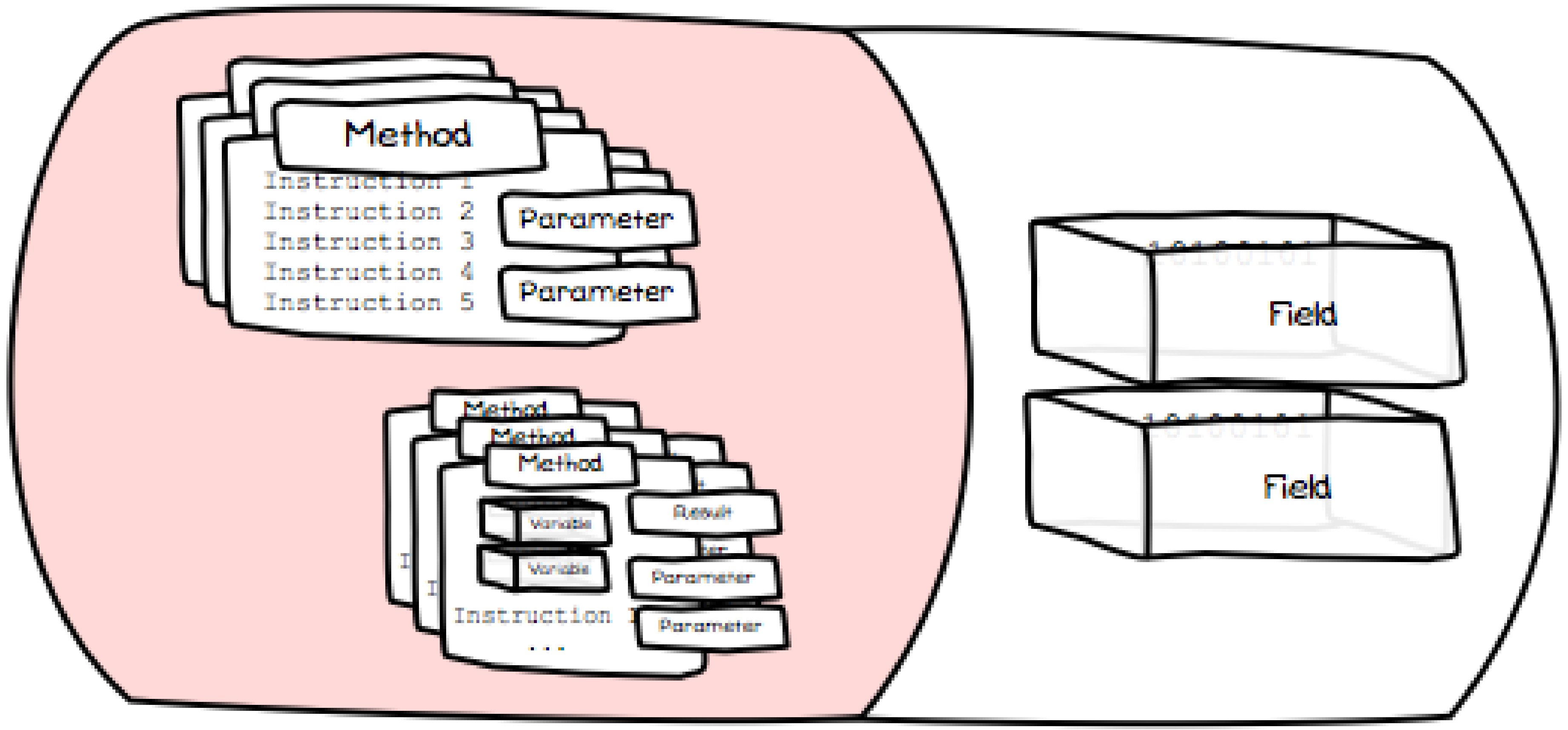
```
class BankAccountPrivate
{
    private string m_name;

    //Declare a CustomerName property of type string
    public string CustomerName
    {
        get { return m_name; }
        set { m_name = value; }
    }
}
```

## Add special methods called **constructors** to initialise your objects when created

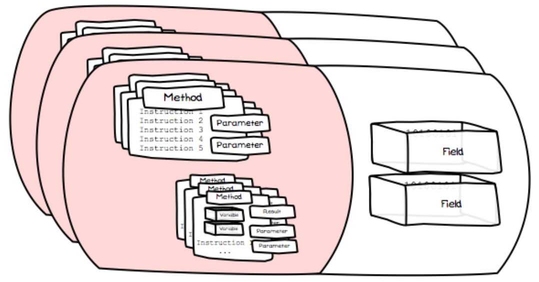


You can get a text representation to print to the Terminal
Will be used to represent a playing card
Use this template to create Card objects
a Card can do ...
a Card 'Knows' ...
Cards can be face up or face down

Constructors are declared within the class



These define how to create/initialise the objects.



Let’s have a look on constructors !

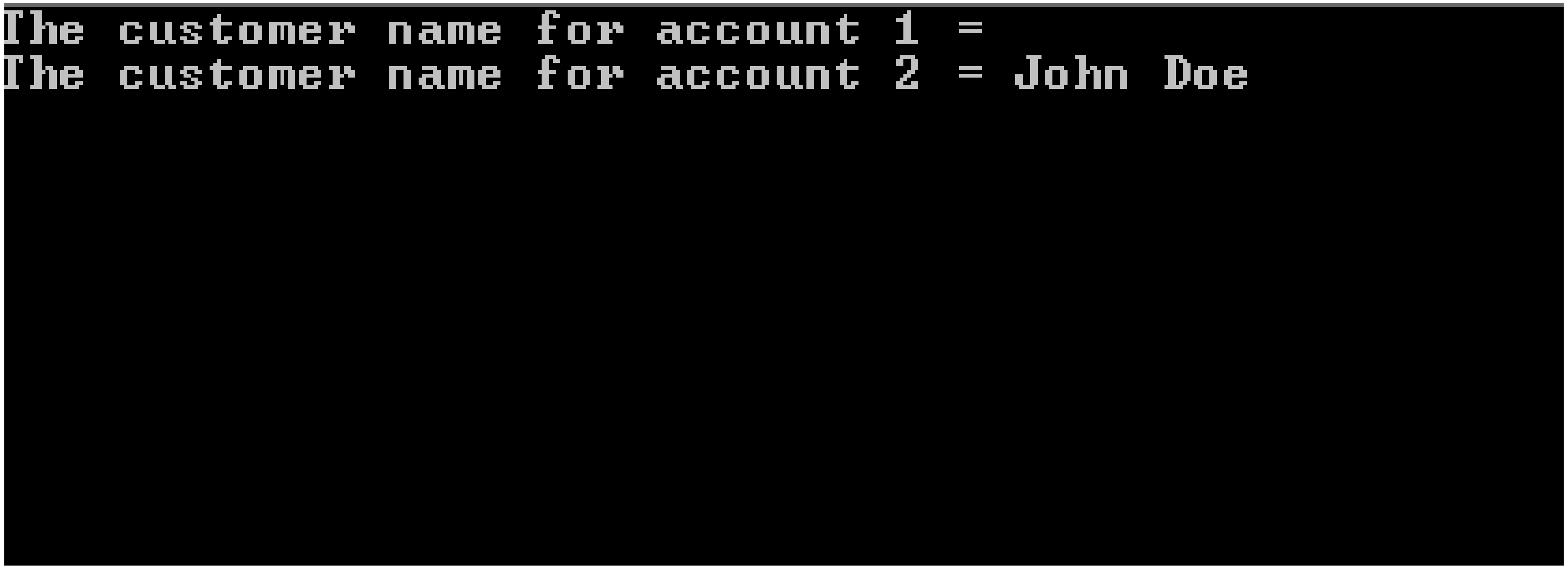
```
public class BankAccountPrivate
{
    private string m_name;

    //Declare a CustomerName property
     of type string
    public string CustomerName
    {
        get { return m_name; }
        set { m_name = value; }
    }

     public BankAccountPrivate(){
          //default constructor
     }
     //pass-by-value constructor
     public BankAccountPrivate(string name){
        m_name = name;
     }
}
```

Two classes: -

- **BankAccountPrivate** – template for creating BankAccountPrivate objects
    - Attributes – m_name
    - Property – CustomerName
    - Default Constructor
    - Pass-by-value constructor
- **MainClass** – where the main program runs

```
class MainClass{
    public static void Main(string[] args){
        BankAccountPrivate account1 = new BankAccountPrivate (); //creating account1 object instance
        Console.WriteLine("The name for account 1 = " + account1.CustomerName);
        BankAccountPrivate account2 = new BankAccountPrivate ("John Doe"); //creating account2 object instance
        Console.WriteLine("The name for account 2 = " + account2.CustomerName);
        Console.ReadLine();
    }
}
```

## Program Output



```
The customer name for account 1 =
The customer name for account 2 = John Doe
```

# Implementing basic classes in C#

<Access Specifier>

<Class Name>

```
public class Customer
{
    //Fields, properties, methods and events are
     added here
}
```

**Creating instances of object**

```
Customer cust1 = new Customer();
Customer cust2 = new Customer();
```

A class constructor is a special member function of a class that is executed whenever we create new objects of that class.

A constructor has exactly the same name as that of class and it does not have any return type

```
public class Customer

   public string name;

   public Customer(string c_name)
    {
         name = c_name;
    }
```

constructor

Invokes the constructor

```
Customer cust1= new Customer("Mary Ann");
Console.WriteLine(cust1.name);
```

# Setting values of ***public*** class variables

```
public class Box
{
       public double length;   // Length of a box
       public double breadth;  // Breadth of a box
       public double height;   // Height of a box
 }

 class MainClass
  {
       public static void Main(string[] args)
       {
           Box Box1 = new Box();   // Declare Box1 of type Box

           // Box 1 specification
           Box1.height = 10.0;
           Box1.length = 8.0;
           Box1.breadth = 5.5;
        }
  }
```

# Setting values of *private* class variables

```
public class Box
{
      private double length;   // Length of a box
      private double breadth;  // Breadth of a box
      private double height;   // Height of a box

       //list of properties
      public double box_length{
           get { return length; }
           set { length = value; }
      }
      //continue for breadth and height….
 }

 class MainClass
   {
         public static void Main(string[] args)
         {
             Box Box1 = new Box();   // Declare Box1 of type Box

             // Box 1 specification
             Box1.box_length = 10.0;
             Box1.box_breadth = 8.0;
             Box1.box_height = 5.5;
           }
   }
```

# This Week's Tutorial Tasks

Pass Task 1 - Hello World

Pass Task 2 - Counter

Pass Task 3 – BankAccount (Assessed Task)

Pass Task 4 - C# Programming Reference Sheet

** To be completed during the tutorial session **