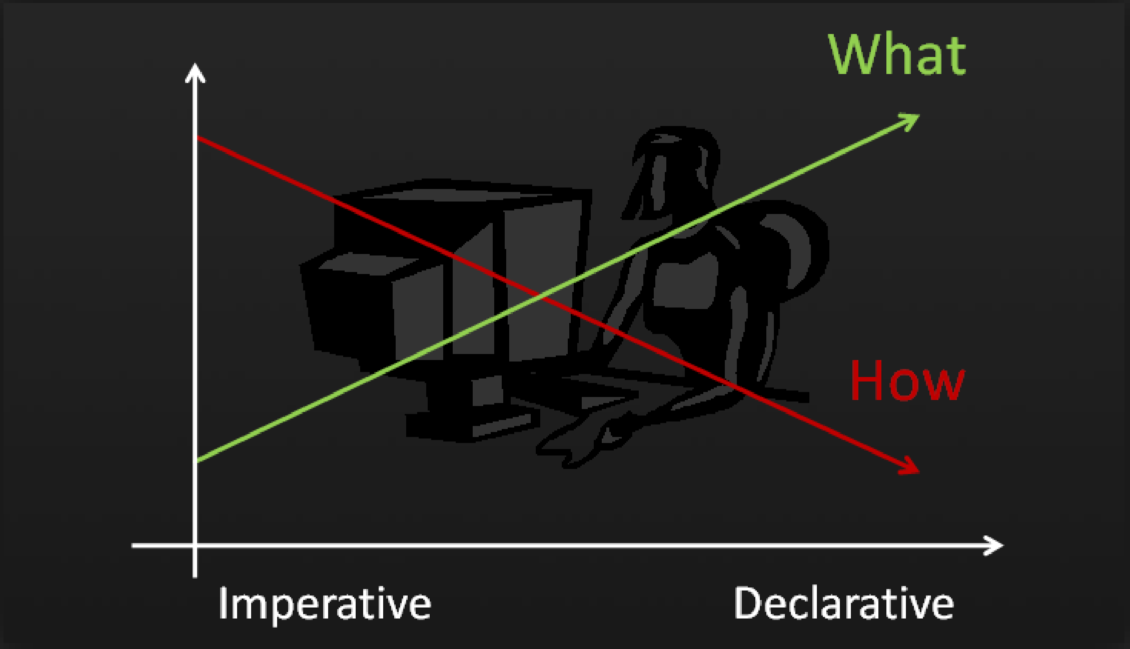
O
O
P
Object
Oriented
Programming

Objectives

- Review the paradigms of programming languages.
- Introduce the language of Object-Oriented programming.
- Provide explicit details for Java implementation of Object-Oriented programming.

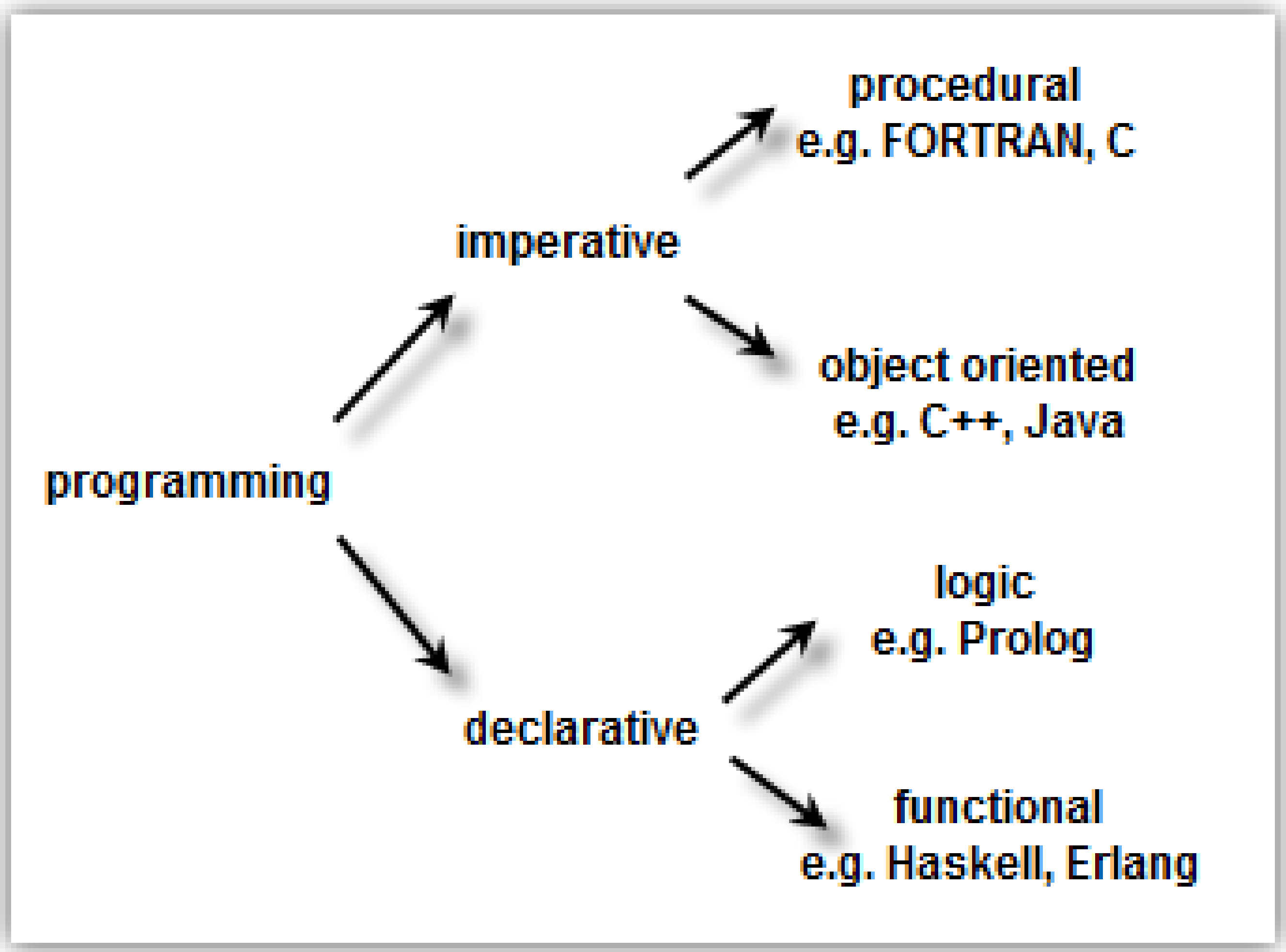
Recall the Programming Paradigms



There are two fundamental **programming paradigms**:

- **Declarative programming** centers on **<u>what</u>** computation should be performed.
- **Imperative programming centers on <u>how</u>** to compute the problem *explicitly*.

Implementation Classification



Imperative

- **Imperative** specifies the sequence of operations to perform in order to achieve a desired result.

Wikipedia

Imperative

- **Imperative** specifies the sequence of operations to perform in order to achieve a desired result.
  - **procedural** uses procedures, such as functions, to describe the commands the computer should perform.

Wikipedia

Imperative

- **Imperative** specifies the sequence of operations to perform in order to achieve a desired result.
  - **procedural** uses procedures, such as functions, to describe the commands the computer should perform.
  - **Object Oriented** represents the concept of objects that have data fields (descriptive attributes of objects) and associated procedures known as methods.

Wikipedia

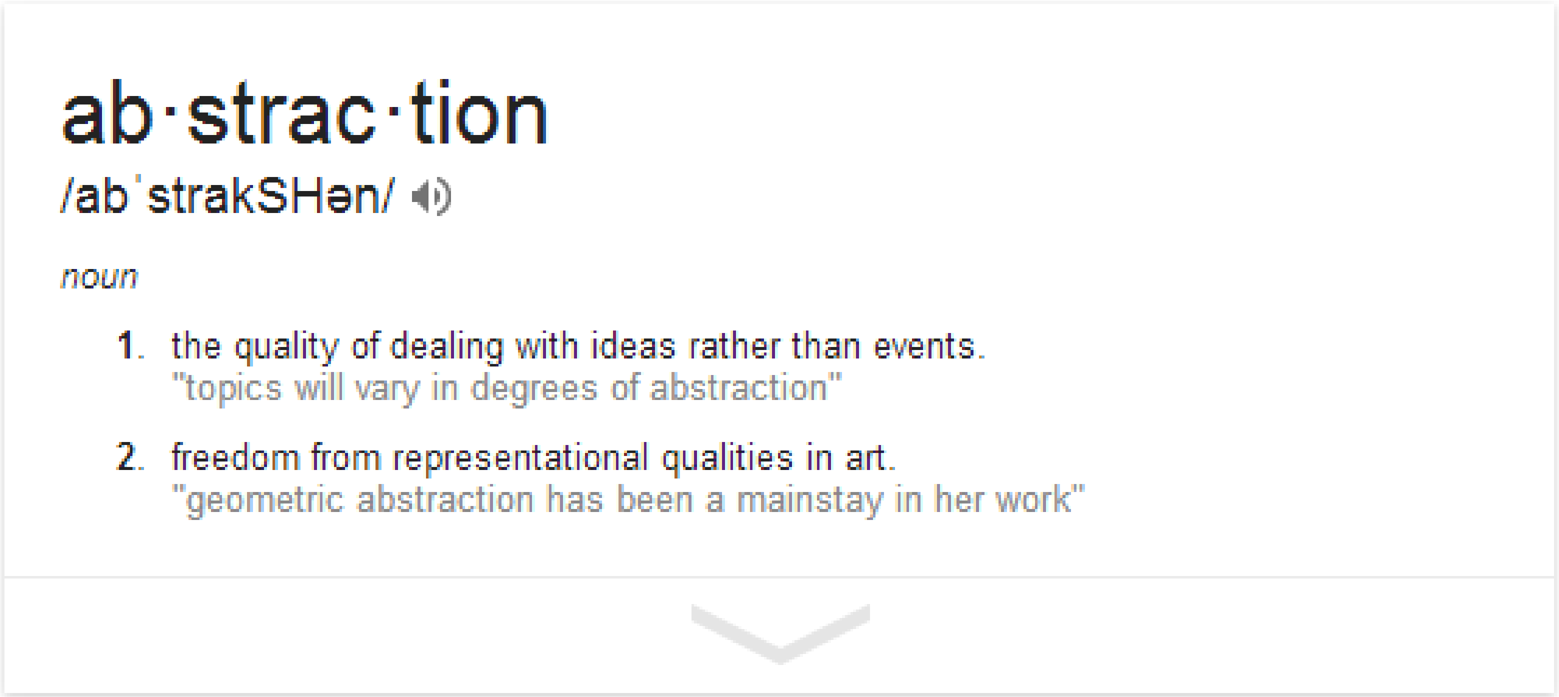
Imperative

- **Imperative** specifies the sequence of operations to perform in order to achieve a desired result.
  - **procedural** uses procedures, such as functions, to describe the commands the computer should perform.
  - **Object Oriented** represents the concept of objects that have data fields (descriptive attributes of objects) and associated procedures known as methods.

In fact, OO should be an extension along the Imperative branch when increasing abstraction is taken into account.

Wikipedia

Abstraction



ab·strac·tion

/abˈstrakSHən/

noun

1. the quality of dealing with ideas rather than events.
   "topics will vary in degrees of abstraction"
2. freedom from representational qualities in art.
   "geometric abstraction has been a mainstay in her work"

Abstraction

- Abstraction hides unnecessary details for some immediate purpose.

Abstraction

- Abstraction hides unnecessary details for some immediate purpose.
- Declarative programming simply ***abstracts*** away the details of *what* to do.

Abstraction

- Abstraction hides unnecessary details for some immediate purpose
- Declarative programming simply *abstracts* away the details of *what* to do.
- We have NOT been doing declarative programming, but will have an overview of it next week from guest lecturer, Professor Warren Burton.

Abstraction

- Abstraction hides unnecessary details for some immediate purpose
- Declarative programming simply *abstracts* away the details of *what* to do.
- We have NOT been doing declarative programming, but will have an overview of it next week from guest lecturer, Professor Warren Burton.
- Similarly, Object-Oriented programming *abstracts away the actions of programming*, allowing the programmer to focus on real-world **objects**.

Definition (*Wikipedia*)

- **Object-oriented programming** (**OOP**):
  A programming [sub-]paradigm that represents the concept of "objects" that have data fields (*attributes* that describe the object) and associated procedures known as *methods*.

Definition (*Wikipedia*)

- **Object-oriented programming (OOP)**: A programming [sub-]paradigm that represents the concept of "objects" that have data fields (*attributes* that describe the object) and associated procedures known as *methods*.
  - Objects, which are usually *instances of classes*, are used to interact with one another to design applications and computer programs.

Definition (*Wikipedia*)

- **Object-oriented programming (OOP)**:

  – Examples are C++, C#, Java, PHP, Perl, and Python.

Definition

- Object oriented programming (OOP) is a programming model where code and data are encapsulated into units called objects that behave semi-autonomously.
- Interaction between objects is arranged through subroutines called methods that are object specific.
- Modularity of objects makes for an easy transferability between different applications.
- Objects are arranged into hierarchies of classes with objects at lower levels inheriting methods from the upper level classes but also adding more specialized ones of their own.
- So classes can be used as object constructors [templates].

Historical Perspective

- Originally, a computer program was viewed as a logical procedure that takes input data, processes it, and produces output data.
- Object-oriented programming (OOP) is a programming language model organized around objects rather than "actions" and data rather than logic.

Historical Perspective

- Early programmers saw the challenge of coding algorithms as how to write the logic, and **not** how to define the data.

Historical Perspective

- Early programmers saw the challenge of coding algorithms as how to write the logic, and **not** how to define the data.
- Object-oriented programming takes the view that **what we really care about are the objects we want to manipulate rather than the logic required to manipulate them**.

Historical Perspective

- Early programmers saw the challenge of coding algorithms as how to write the logic, and **not** how to define the data.
- Object-oriented programming takes the view that **what we really care about are the objects we want to manipulate rather than the logic required to manipulate them**.
  - Examples of objects range from human beings (described by name, address, and so forth) to buildings and floors (whose properties can be described and managed) down to the little widgets on a computer desktop (such as buttons and scroll bars).

Historical Perspective

- The object-oriented approach is considered a logical extension of good design practices that go back to the very beginning of computer programming rather than being revolutionary as claimed by some proponents.

Historical Perspective

- The object-oriented approach is considered a logical extension of good design practices that go back to the very beginning of computer programming rather than being revolutionary as claimed by some proponents.
- Object-oriented programming is merely a logical extension of older techniques such as *structured programming* and *Abstract Data Structures* (**An object is an Abstract Data Structure with the addition of *polymorphism* and *inheritance***, *to be defined shortly*).

Introductory Considerations

- Rather than structure programs as code and data, an object-oriented system integrates the two using the concept of an "object".

Introductory Considerations

- Rather than structure programs as code and data, an object-oriented system integrates the two using the concept of an "object".
- An object has state (data) and behavior (code).

Introductory Considerations

- Rather than structure programs as code and data, an object-oriented system integrates the two using the concept of an "object".
- An object has state (data) and behavior (code).
- Objects correspond to things found in the real world.

  For example,

  – a graphics program will have objects such as *circle*, *square*, *menu*.

  – An online shopping system will have objects such as *shopping cart*, *customer*, *product*. The shopping system will support behaviors such as *place order*, *make payment*, and *offer discount*.

Introductory Considerations

- Rather than structure programs as code and data, an object-oriented system integrates the two using the concept of an "object".
- An object has state (data) and behavior (code).
- Objects correspond to things found in the real world.

  For example,
  - a graphics program will have objects such as *circle*, *square*, *menu*.
  - An online shopping system will have objects such as *shopping cart*, *customer*, *product*. The shopping system will support behaviors such as *place order*, *make payment*, and *offer discount*.
- The objects are designed as class hierarchies.

  For example,
  - with the shopping system there might be high level classes such as *electronics product*, *kitchen product*, and *book*.
  - There may be further refinements for example under *electronic products*: *Personal Computer, Cell phone, Phased plasma rifle in the 40W range*, *etc.*

Introductory Considerations

- Rather than structure programs as code and data, an object-oriented system integrates the two using the concept of an "object".
- An object has state (data) and behavior (code).
- Objects correspond to things found in the real world.

  For example,
  - a graphics program will have objects such as *circle*, *square*, *menu*.
  - An online shopping system will have objects such as *shopping cart*, *customer*, *product*. The shopping system will support behaviors such as *place order*, *make payment*, and *offer discount*.
- The objects are designed as class hierarchies.

  For example,
  - with the shopping system there might be high level classes such as *electronics product*, *kitchen product*, and *book*.
  - There may be further refinements for example under *electronic products*: *Personal Computer, Cell phone, Phased plasma rifle in the 40W range, etc.*
- **These classes and subclasses correspond to sets and subsets in mathematical logic.**

Data Modeling

- The first step in OOP is to identify all the objects the programmer wants to manipulate and how they relate to each other, an exercise often known as **data modeling**.
- Once an object has been identified, it is generalized as a **class of objects** (think of Plato's concept of the "ideal" chair that stands for all chairs) which defines the kind of data it contains and any logic sequences that can manipulate it.
- Each distinct logic sequence is known as a **method**.
- Objects communicate with well-defined interfaces called **messages**.

http://searchsoa.techtarget.com/definition/object-oriented-programming

Benefits of OO Programming

The concepts and rules used in object-oriented programming provide these important benefits:

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics.
  Called **inheritance**, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.
- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of **data hiding (*a.k.a.* encapsulation)** provides greater system security and avoids unintended data corruption.
- The definition of a class is reusable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).
- The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

http://searchsoa.techtarget.com/definition/object-oriented-programming

Communication/Synchronization

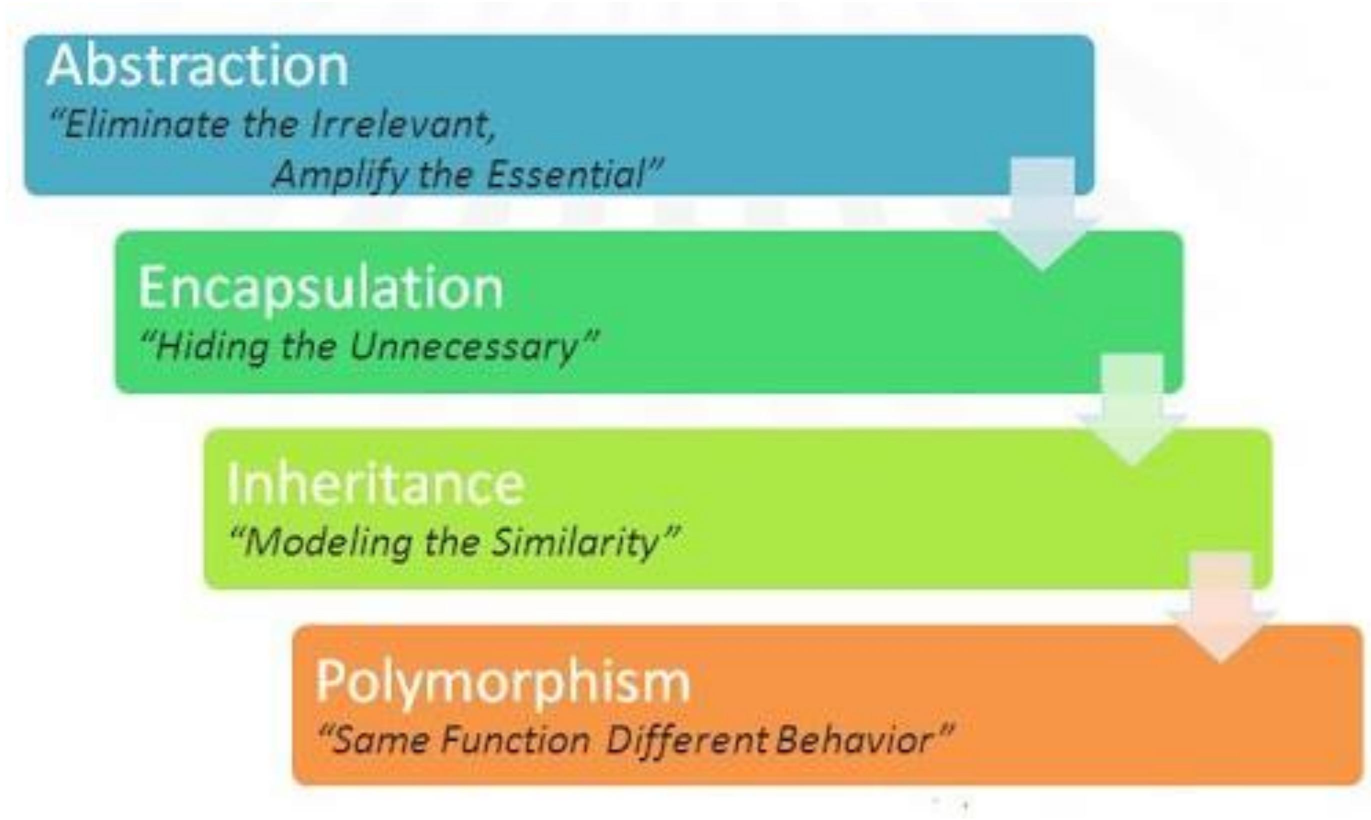
- Sometimes objects need to communicate with one another.
- This is done through Object Oriented Protocols:
  - Definition: OO Protocol
    - In object-oriented programming, a **protocol** or **interface** is a common means for unrelated objects to communicate with each other. These are definitions of methods and values which the objects agree upon in order to cooperate.
    - See http://en.wikipedia.org/wiki/Protocol_%28object-oriented_programming%29

WIKIPEDIA
The Free Encyclopedia

Let Us Review

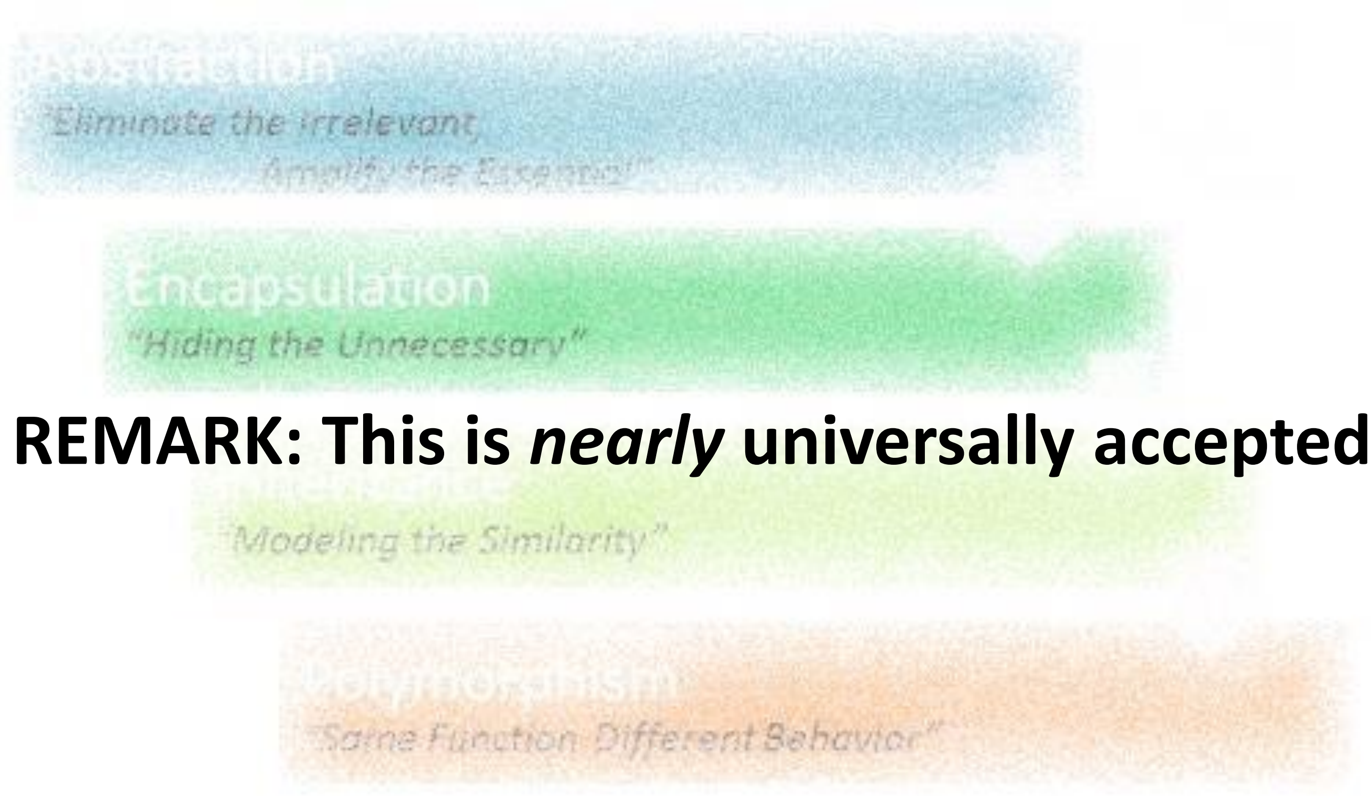
Primary Characteristics of OO

Object-oriented methodology relies on four main characteristics that define object-oriented languages:


Abstraction
"Eliminate the Irrelevant, Amplify the Essential"
Encapsulation
"Hiding the Unnecessary"
Inheritance
"Modeling the Similarity"
Polymorphism
"Same Function Different Behavior"

Primary Characteristics of OO

Object-oriented methodology relies on four main characteristics that define object-oriented languages:


Abstraction
"Eliminate the irrelevant,
Amplify the Essential"
Encapsulation
"Hiding the Unnecessary"
"Modeling the Similarity"
Polymorphism
"Same Function Different Behavior"

REMARK: This is *nearly* universally accepted.

Terminology

Class

In object-oriented programming, a **class** is an extensible **program-code-template** for creating objects, providing initial values for state (member variables) and implementations of behavior (member functions, methods).

– *Wikipedia*

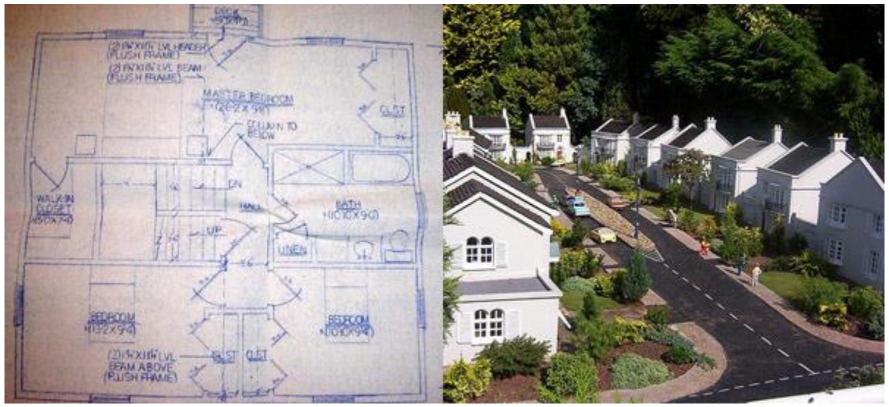
Terminology

*"Object-oriented programming is a style of coding that allows developers to group similar tasks into **classes**."*

Terminology

Class Versus Object

- A class, for example, is like **a blueprint for a house** [template]. It defines the shape of the house on paper, with relationships between the different parts of the house clearly defined and planned out, even though the house doesn't exist.

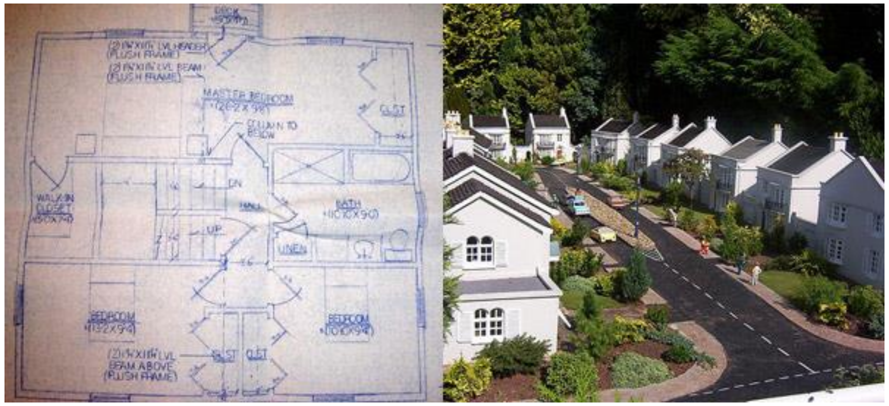


http://code.tutsplus.com/tutorials/object-oriented-php-for-beginners--net-12762

Terminology

Class Versus Object

- A class, for example, is like **a blueprint for a house**. It defines the shape of the house on paper, with relationships between the different parts of the house clearly defined and planned out, even though the house doesn't exist.
- An object, then, is like **the actual house** built according to that blueprint. The data stored in the object is like the wood, wires, and concrete that compose the house: without being assembled according to the blueprint, it's just a pile of stuff. However, when it all comes together, it becomes an organized, useful house.



Terminology

Objects and Methods

- An ***object*** is an encapsulation of data together with procedures that manipulate the data and functions that return information about the data.
- The procedures and functions are both called ***methods***.
- In other words, an *object* is an “encapsulation” of data together with procedures that manipulate the data and functions that return information about the data, and the procedures and functions are both called methods.

Terminology

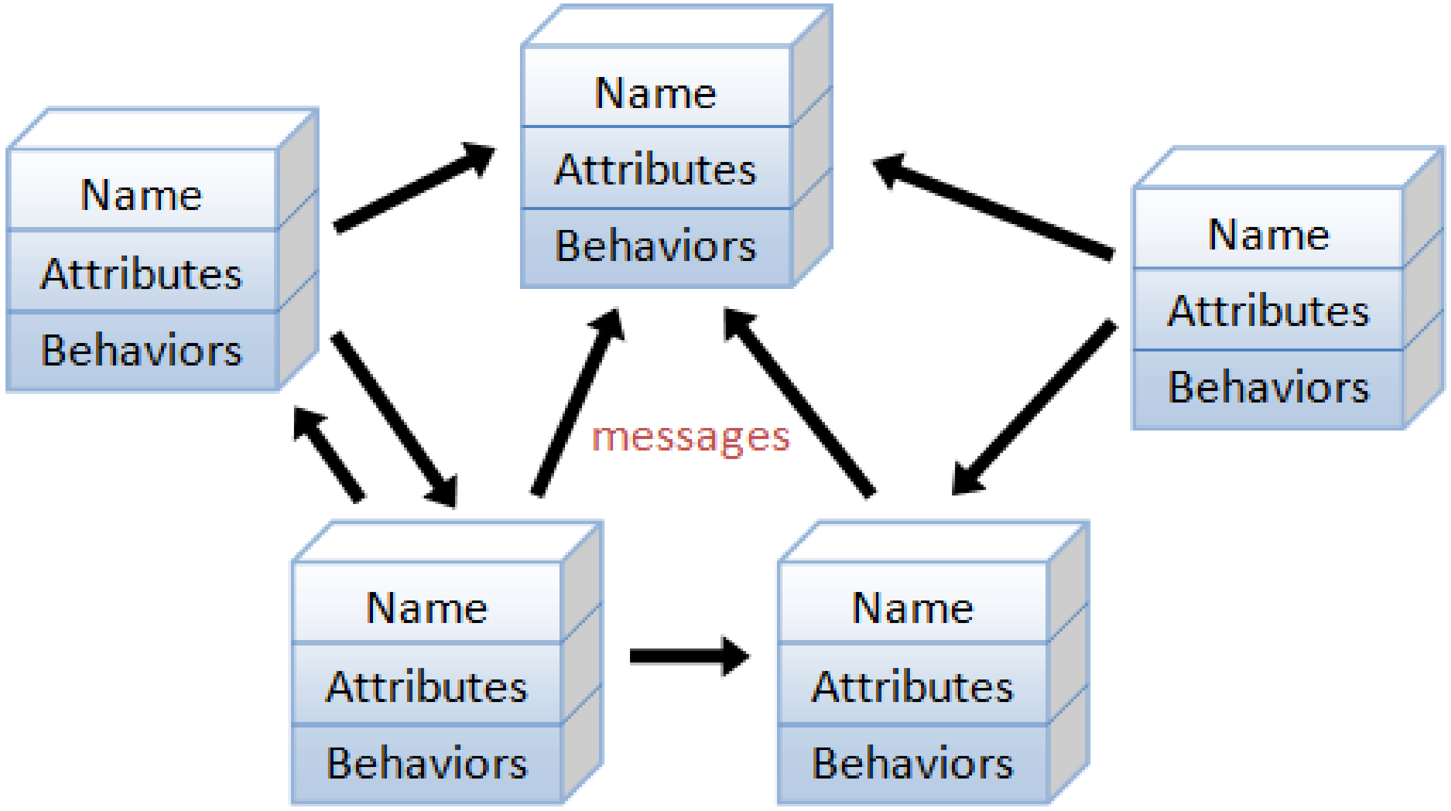
Encapsulation

- *Encapsulation* refers to *abstractions* that allow each object to have its own data and methods.
- Data within an object cannot be accessed directly and this is known as *data encapsulation* where the data and functions are encapsulated in an object.
- The idea of encapsulating data together with methods existed before object-oriented languages were developed.
  - It is, for example, inherent in the concept of an **abstract data type**.

Terminology

Messages and Receivers

In object-oriented languages, encapsulation is effected, in part, by having all method calls handled by objects that recognize the method.



An object-oriented program consists of many well-encapsulated objects and interacting with each other by sending messages

Terminology

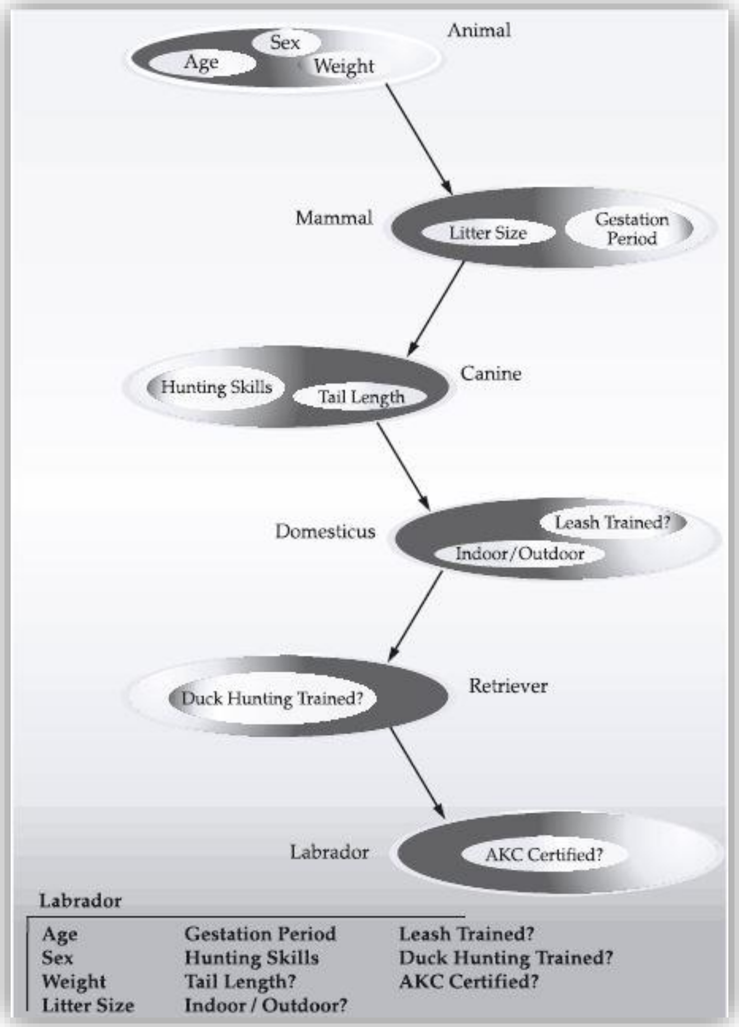
Polymorphism

Polymorphism refers to the capability of having methods with the same names and parameter types exhibit different behavior depending on the receiver.

Terminology

NOTE:

The word ‘poly’ means many and ‘morph’ means form. So polymorphism occurs when a thing possesses many forms (*i.e.,* a single interface with multiple forms).

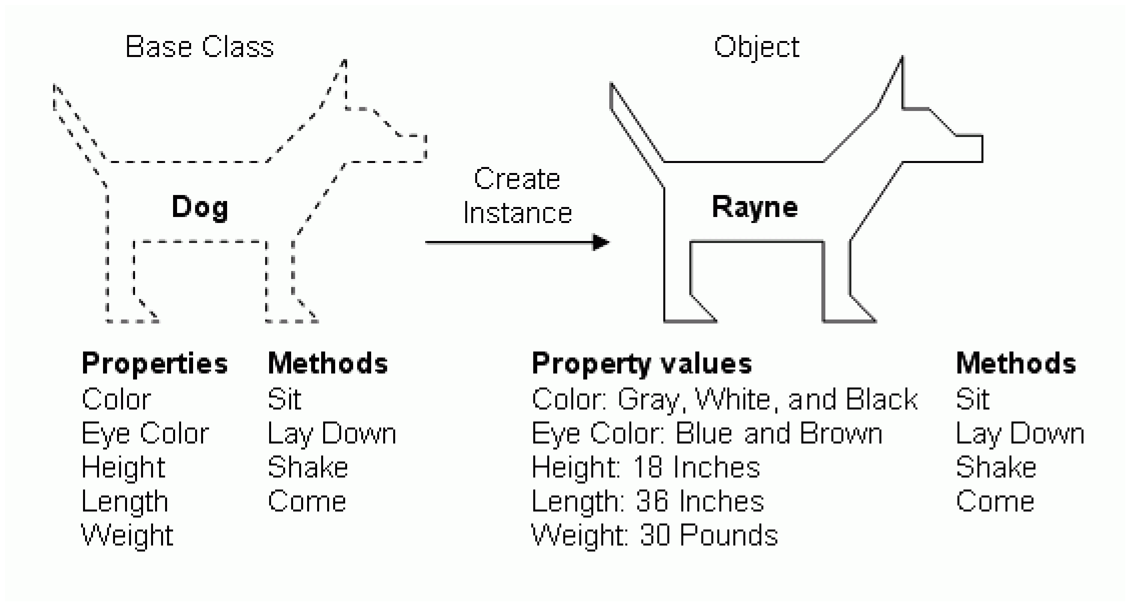


http://www.universocomputacao.com/blog/2013/08/13/object-oriented-thinking/

Terminology

Class

Class is basically a blueprint or template for defining similar objects (objects belong to a class).



Properties	Methods
Color	Sit
Eye Color	Lay Down
Height	Shake
Length	Come
Weight	

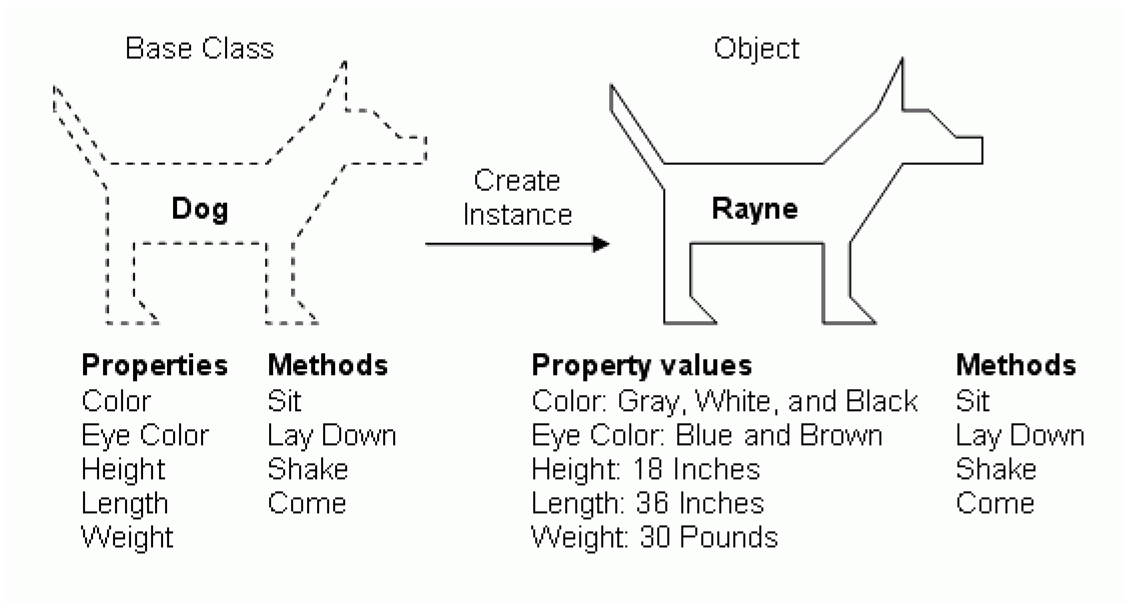
Property values	Methods
Color: Gray, White, and Black	Sit
Eye Color: Blue and Brown	Lay Down
Height: 18 Inches	Shake
Length: 36 Inches	Come
Weight: 30 Pounds	

http://www.universocomputacao.com/blog/2013/08/13/object-oriented-thinking/

Terminology

Inheritance

The concept of grouping classes into subclasses where the subclass derives properties from its parent class.



http://www.universocomputacao.com/blog/2013/08/13/object-oriented-thinking/

Terminology

Data Abstraction

- Abstraction is the process of recognizing and focusing on important characteristics of a situation or object and leaving/filtering out the unwanted characteristics of that situation or object.
- For example, consider a **person** and see how that person is abstracted in various situations:
  - A doctor sees (abstracts) the **person** as patient. The doctor is interested in name, height, weight, age, blood group, previous or existing diseases, *etc*., of a person.
  - An employer sees (abstracts) a **person** as an employee. The employer is interested in name, age, health, degree of study, work experience, *etc*., of a person.
- The common element is *generalization.*

Terminology

Information Hiding

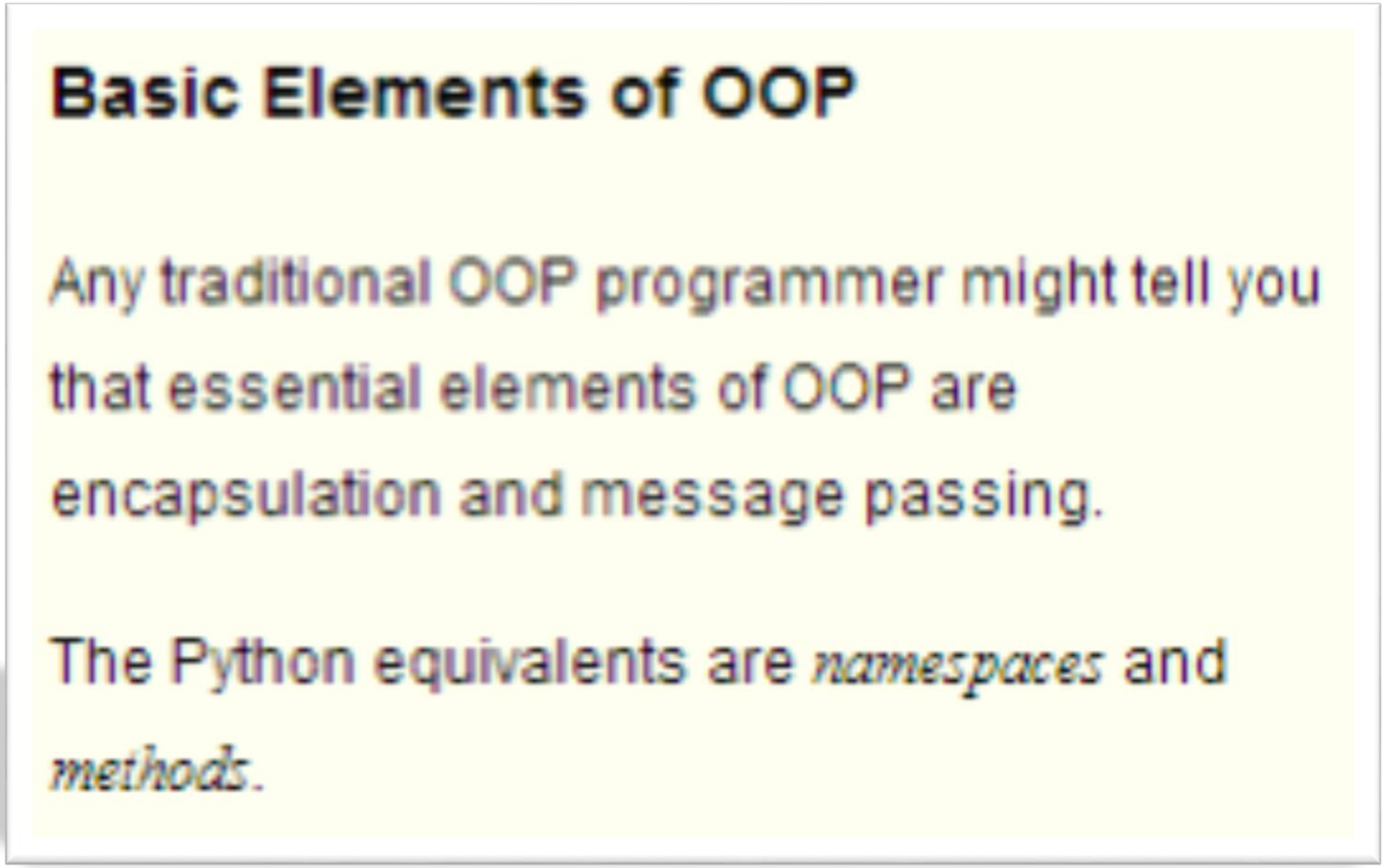
- Generally speaking, this is a synonym for *encapsulation*.
- Not all researchers agree, however; one may think of *information hiding as being the principle* and *encapsulation being the technique*.
- A software module hides information by encapsulating the information into a module or other construct which presents an *interface.*

Encapsulation Versus Data/Information Hiding

- Data and information hiding are a broader notions, found in computer science and software engineering. It refers to the fact that those part of a computer program that may change must not be accessible from other modules/from clients.
- Encapsulation is a term that is found in Object-Oriented paradigm and refers to keeping the data in private fields and modify it only through methods.
- Thus, **encapsulation may be seen as a way of achieving data hiding in object-oriented systems**.

http://programmers.stackexchange.com/questions/173547/what-is-the-difference-between-data-hiding-and-encapsulation

AND NOW, JUST TO CONFUSE YOU…



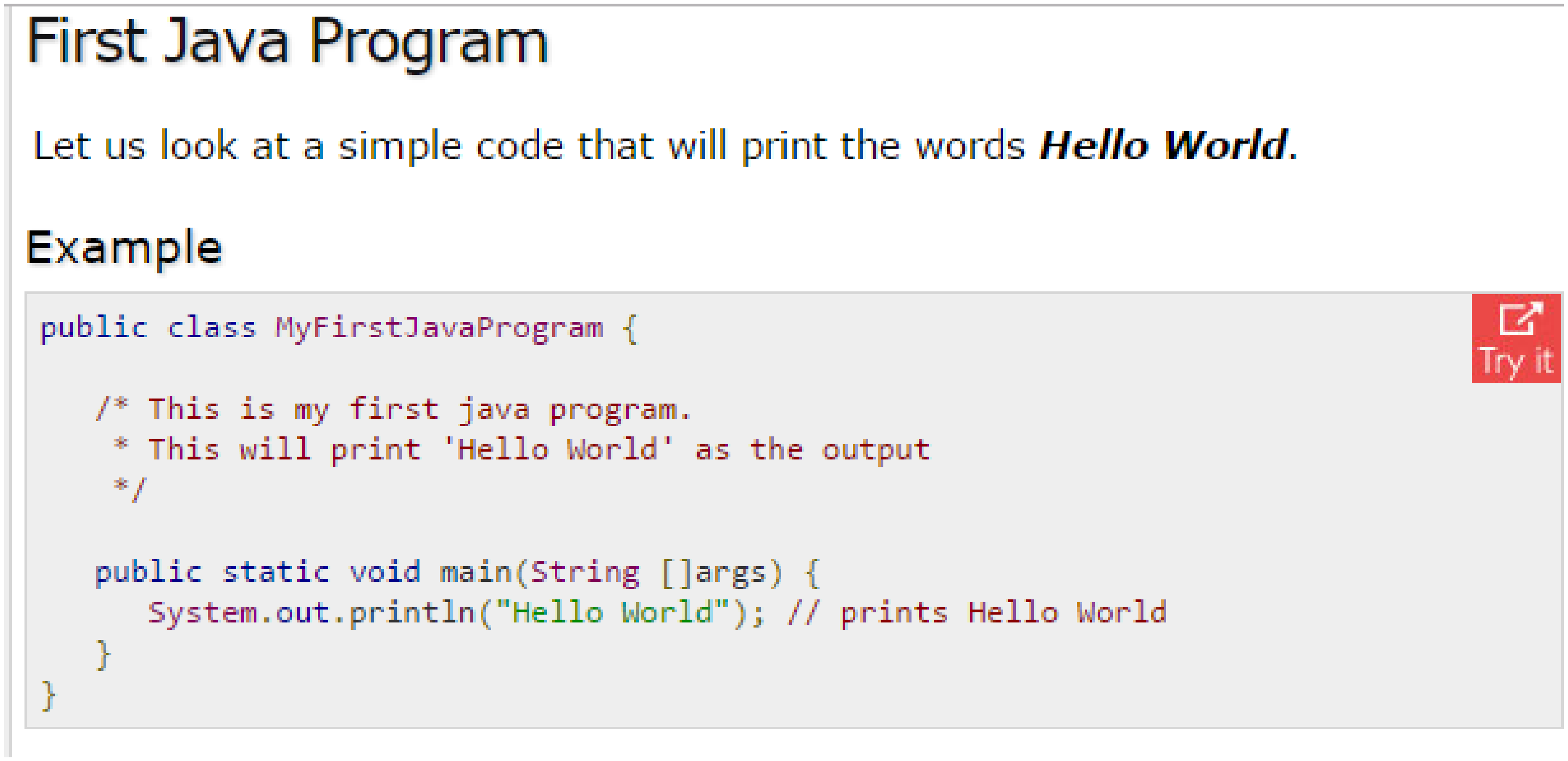
Basic Elements of OOP

Any traditional OOP programmer might tell you that essential elements of OOP are encapsulation and message passing.

The Python equivalents are *namespaces* and *methods*.

http://www.voidspace.org.uk/python/articles/OOP.shtml

Basic Review of Java



First Java Program

Let us look at a simple code that will print the words ***Hello World***.

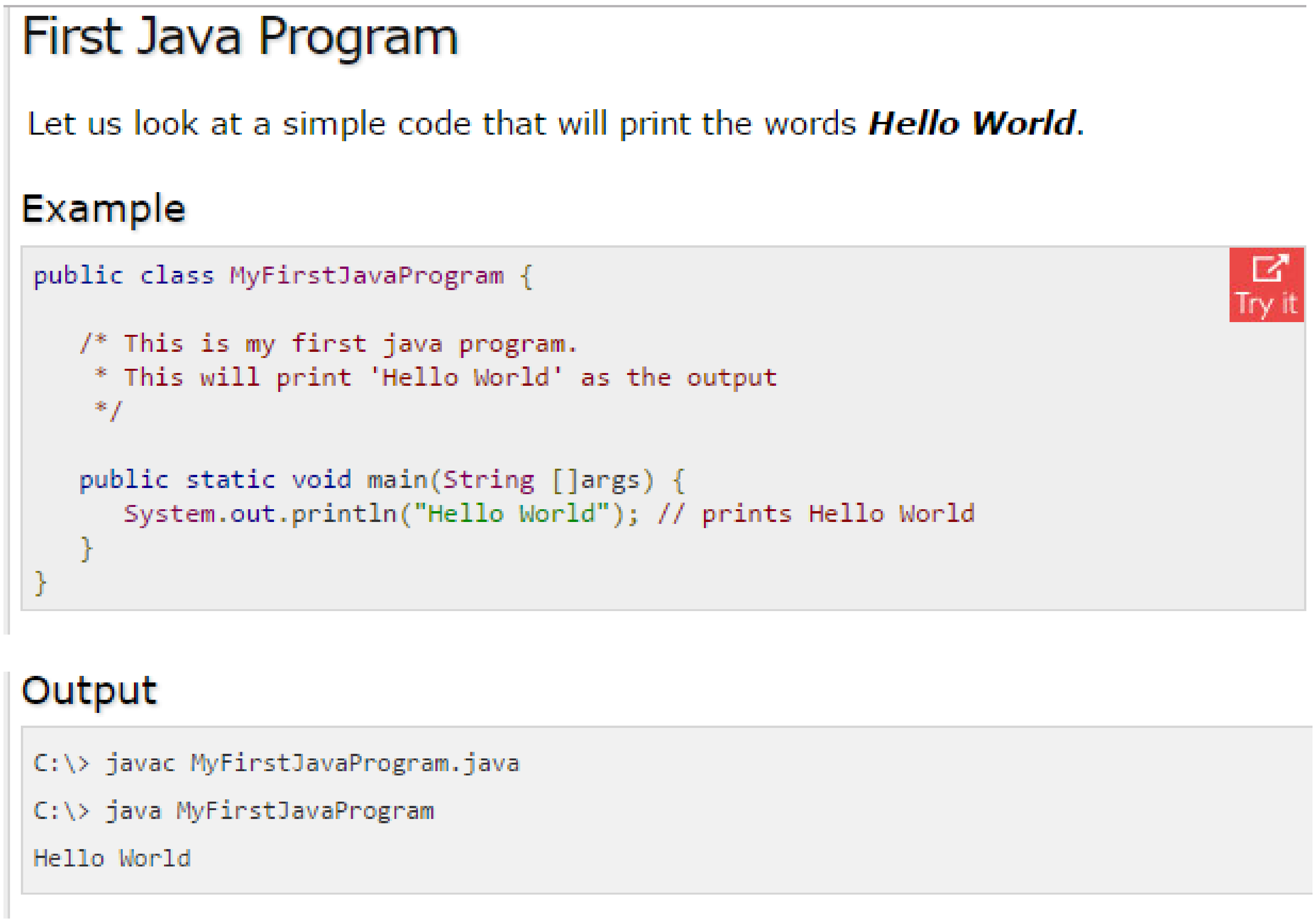
Example

```
public class MyFirstJavaProgram {

   /* This is my first java program.
    * This will print 'Hello World' as the output
    */

   public static void main(String []args) {
      System.out.println("Hello World"); // prints Hello World
   }
}
```

https://www.tutorialspoint.com/java/java_basic_syntax.htm



First Java Program

Let us look at a simple code that will print the words ***Hello World***.

Example

```
public class MyFirstJavaProgram {

   /* This is my first java program.
    * This will print 'Hello World' as the output
    */

   public static void main(String []args) {
      System.out.println("Hello World"); // prints Hello World
   }
}
```

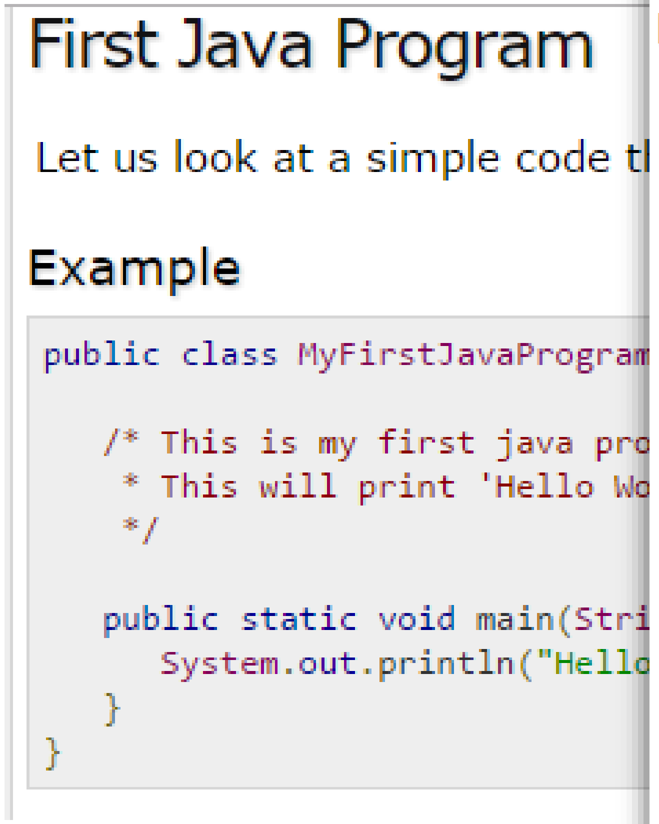
Output

```
C:\> javac MyFirstJavaProgram.java

C:\> java MyFirstJavaProgram

Hello World
```

https://www.tutorialspoint.com/java/java_basic_syntax.htm



First Java Program

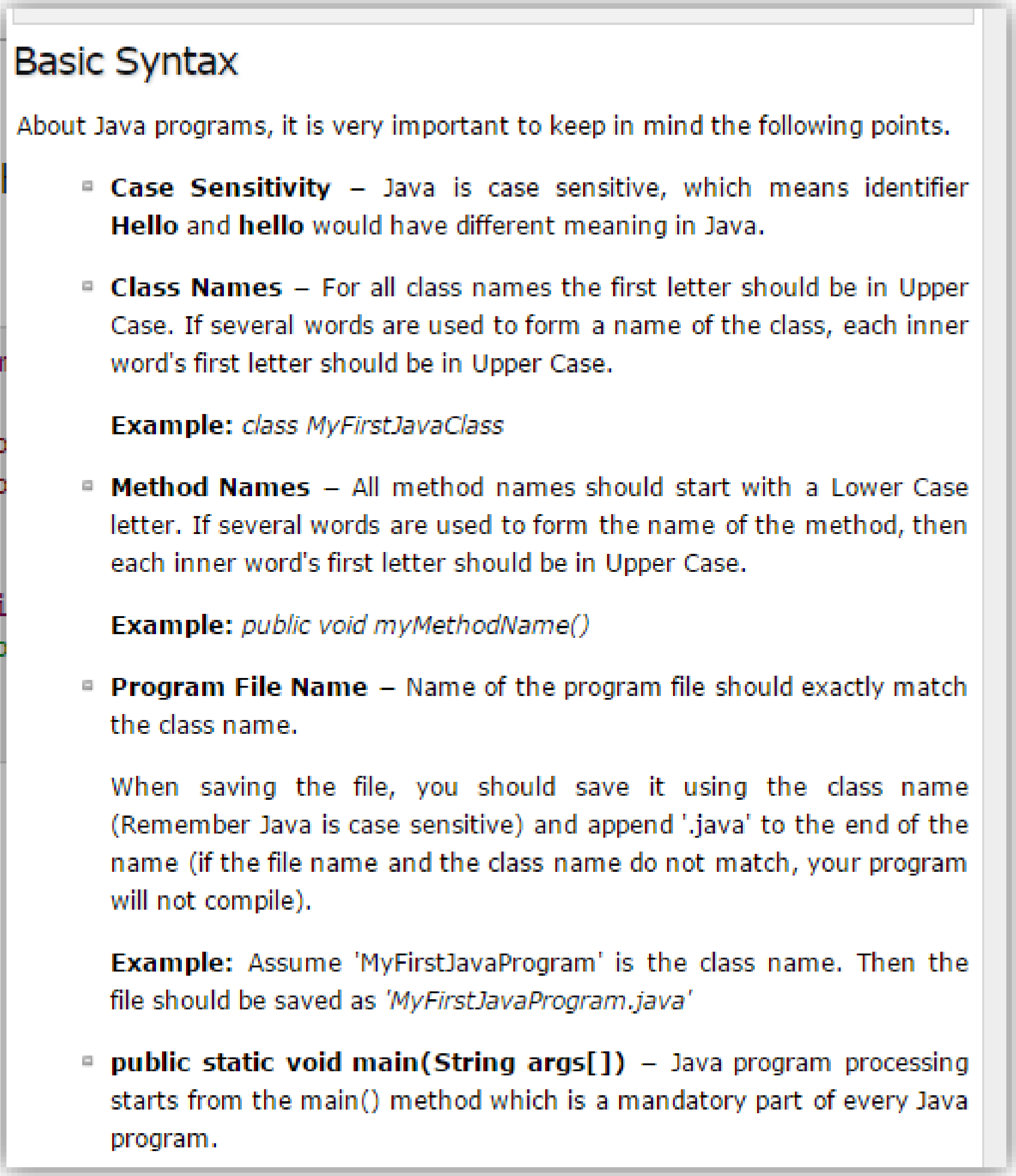
Let us look at a simple code tl

Example

```
public class MyFirstJavaProgram

   /* This is my first java pro
    * This will print 'Hello Wo
    */

   public static void main(Stri
      System.out.println("Hello
   }
}
```



Basic Syntax

About Java programs, it is very important to keep in mind the following points.

- **Case Sensitivity** – Java is case sensitive, which means identifier **Hello** and **hello** would have different meaning in Java.
- **Class Names** – For all class names the first letter should be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.

  Example: *class MyFirstJavaClass*
- **Method Names** – All method names should start with a Lower Case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.

  Example: *public void myMethodName()*
- **Program File Name** – Name of the program file should exactly match the class name.

  When saving the file, you should save it using the class name (Remember Java is case sensitive) and append '.java' to the end of the name (if the file name and the class name do not match, your program will not compile).

  Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as *'MyFirstJavaProgram.java'*
- **public static void main(String args[])** – Java program processing starts from the main() method which is a mandatory part of every Java program.

https://www.tutorialspoint.com/java/java_basic_syntax.htm

OO Aspects of Java

Taken from http://faculty.kfupm.edu.sa/ICS/saquib/ICS202/Unit01_Review.pdf

OO Aspects of Java

- Java provides explicit support for many of the fundamental Object - Oriented Concepts.
- For Java, the three most important are:
  - **Encapsulation**: Representing data and the set of operations on the data as a single entity - exactly what classes do.
  - **Inheritance**: Java allows related classes to be organized in a hierarchical manner using the extends keyword.
  - **Polymorphism**: Same code behaves differently at different times during execution. This is due to dynamic binding.
- Also, it is important to consider:
  - Classification: Grouping related things together. This is supported through classes, inheritance & packages.
  - Information Hiding: An object should be in full control of its data, granting specific access only to whom it wishes.

Review of inheritance

- Suppose we have the following **Employee** class:

```
class Employee  {
   protected String name;
   protected double payRate;
   public Employee(String name, double payRate)  {
      this.name = name;
      this.payRate = payRate;
   }
   public String getName()  {return name;}
   public void setPayRate(double newRate)  {
      payRate = newRate;
   }
   public double pay()  {return payRate;}
   public void print()  {
      System.out.println("Name: " + name);
      System.out.println("Pay Rate: "+payRate);
   }
}
```

Review of inheritance (contd.)

- Now, suppose we wish to define another class to represent a part-time employee whose salary is paid per hour. We inherit from the **Employee** class as follows:

```
class HourlyEmployee extends Employee  {
   private int hours;
   public HourlyEmployee(String hName, double hRate)  {
      super(hName, hRate);
      hours = 0;
   }
   public void addHours(int moreHours)  {hours += moreHours;}
   public double pay()  {return payRate * hours;}
   public void print()  {
      super.print();
      System.out.println("Current hours: " + hours);
   }
}
```

Notes about Inheritance

- We observe the following from the examples on inheritance:
  - Methods and instance variables of the super class are inherited by subclasses, thus allowing for code reuse.
  - A subclass can define additional instance variables (e.g. hours) and additional methods (e.g. addHours).
  - A subclass can override some of the methods of the super class to make them behave differently (e.g. the pay & print)
  - Constructors are not inherited, but can be called using the super keyword. such a call must be the first statement.
  - If the constructor of the super class is not called, then the complier inserts a call to the default constructor -watch out!
  - super may also be used to call a method of the super class.

Review of Abstract Classes

- Inheritance enforces hierarchical organization, the benefit of which are: reusability, type sharing and polymorphism.
- Java uses Abstract classes & Interfaces to further strengthen the idea of inheritance.
- To see the role of abstract of classes, suppose that the **pay** method is not implemented in the **HourlyEmployee** subclass.
- Obviously, the **pay** method in the **Employee** class will be assumed, which will lead to wrong result.
- One solution is to remove the **pay** method out and put it in another extension of the Employee class, **MonthlyEmployee.**
- The problem with this solution is that it does not force subclasses of **Employee** class to implement the **pay** method.

Review of Abstract Classes (Cont'd)

- The solution is to declare the pay method of the Employee class as abstract, thus, making the class abstract.

```
abstract class Employee  {
   protected String name;
   protected double payRate;
   public Employee(String empName, double empRate)  {
     name = empName;
     payRate = empRate;
   }
   public String getName()  {return name;}
   public void setPayRate(double newRate)  {payRate = newRate;}

   abstract public double pay();

   public void print()  {
     System.out.println("Name: " + name);
     System.out.println("Pay Rate: "+payRate);
   }
 }
```

Review of Abstract Classes (Cont'd)

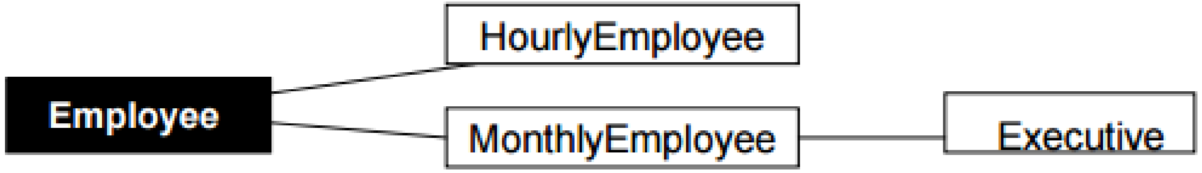
- The following extends the Employee abstract class to get MonthlyEmployee class.

```
class MonthlyEmployee extends Employee  {
   public MonthlyEmployee(String empName, double empRate) {
       super(empName, empRate);
   }
   public double pay()  {
       return payRate;
   }
}
```

- The next example extends the MonthlyEmployee class to get the Executive class.

Review of Abstract Classes (Cont'd)

```
class Executive extends MonthlyEmployee  {
    private double bonus;
    public Executive(String exName, double exRate)  {
        super(exName, exRate);
        bonus = 0;
    }
    public void awardBonus(double amount)  {
        bonus = amount;
    }
    public double pay()  {
        double paycheck = super.pay() + bonus;
        bonus = 0;
        return paycheck;
    }
    public void print()  {
        super.print();
        System.out.println("Current bonus: " + bonus);
    }
}
```

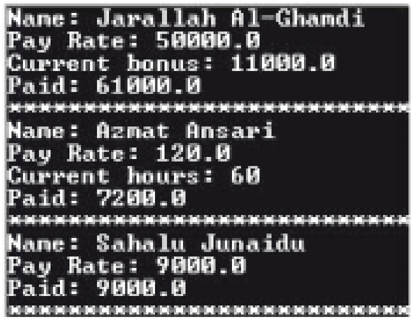


Review of Abstract Classes (Cont'd)

- The following further illustrates the advantages of organizing classes using inheritance - same type, polymorphism, etc.

```java
public class TestAbstractClass  {
   public static void main(String[] args)  {
     Employee[] list = new Employee[3];
     list[0] = new Executive("Jarallah Al-Ghamdi", 50000);
     list[1] = new HourlyEmployee("Azmat Ansari", 120);
     list[2] = new MonthlyEmployee("Sahalu Junaidu", 9000);
      ((Executive)list[0]).awardBonus(11000);
     for(int i = 0; i < list.length; i++)
        if(list[i] instanceof HourlyEmployee)
           ((HourlyEmployee)list[i]).addHours(60);
     for(int i = 0; i < list.length; i++)  {
         list[i].print();
          System.out.println("Paid: " + list[i].pay());
          System.out.println("**************************");
     }
   }
}
```

The Program Output



```
Name: Jarallah Al-Ghamdi
Pay Rate: 50000.0
Current bonus: 11000.0
Paid: 61000.0
**************************
Name: Azmat Ansari
Pay Rate: 120.0
Current hours: 60
Paid: 7200.0
**************************
Name: Sahalu Junaidu
Pay Rate: 9000.0
Paid: 9000.0
**************************
```

Review of Interfaces

- Interfaces are not classes, they are entirely a separate entity.
- They provide a list of abstract methods which MUST be implemented by a class that implements the interface.
- Unlike abstract classes which may contain implementation of some of the methods, interfaces provide NO implementation.
- Like abstract classes, the purpose of interfaces is to provide organizational structure.
- More importantly, interfaces are here to provide a kind of "multiple inheritance" which is not supported in Java.
- If both parents of a child implement a method, which one does the child inherits? - Multiple inheritance confusion.
- Interfaces allow a child to be both of type A and B.

Review of Interfaces (contd.)

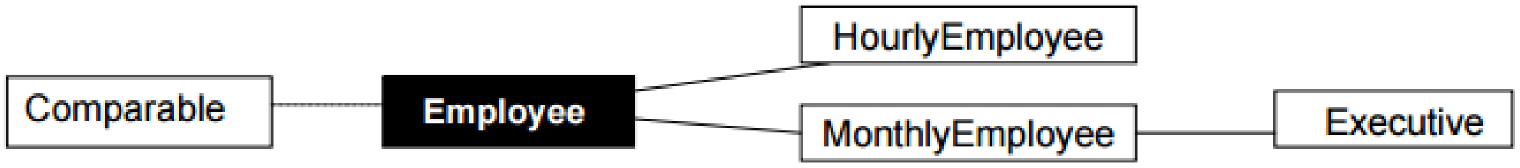
- Recall that Java has the Comparable interface defined as:

```
interface Comparable {
    int compareTo(Object o);
}
```

- Recall also that java has the java.util.Arrays class, which has a sort method that can sort any array whose contents are either primitive values or Comparable objects.
- Thus, to sort our list of Employee objects, all we need is to modify the Employee class to implement the Comparable interface.
- Notice that this will work even if the Employee class is extending another class or implementing another interface.
- This modification is shown in the next page.

Review of Interfaces (contd.)

```
abstract class Employee implements Comparable  {
   protected String name;
   protected double payRate;
   public Employee(String empName, double empRate)  {
        name = empName;
        payRate = empRate;
   }
   public String getName()  {return name;}
   public void setPayRate(double newRate)  {
      payRate = newRate;
   }
   abstract public double pay();
   public int compareTo(Object o)  {
        Employee e = (Employee) o;
        return name.compareTo( e.getName());
   }
}
```

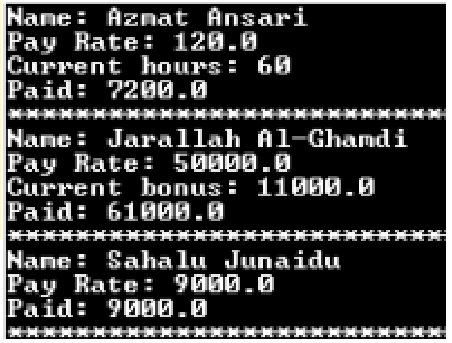


Review of Interfaces (contd.)

- Since Employee class implements the Comparable interface, the array of employees can now be sorted as shown below:

```
import java.util.Arrays;
public class TestInterface  {
   public static void main(String[] args)  {
      Employee[] list = new Employee[3];
      list[0] = new Executive("Jarallah Al-Ghamdi", 50000);
      list[1] = new HourlyEmployee("Azmat Ansari", 120);
      list[2] = new MonthlyEmployee("Sahalu Junaidu", 9000);
      ((Executive)list[0]).awardBonus(11000);
      for(int i = 0; i < list.length; i++)
         if(list[i] instanceof HourlyEmployee)
            ((HourlyEmployee)list[i]).addHours(60);
      Arrays.sort(list);
      for(int i = 0; i < list.length; i++)  {
         list[i].print();
         System.out.println("Paid: " + list[i].pay());
         System.out.println("**********************");
      }
   }
}
```

The program output



```
Name: Azmat Ansari
Pay Rate: 120.0
Current hours: 60
Paid: 7200.0
**************************
Name: Jarallah Al-Ghamdi
Pay Rate: 50000.0
Current bonus: 11000.0
Paid: 61000.0
**************************
Name: Sahalu Junaidu
Pay Rate: 9000.0
Paid: 9000.0
**************************
```

Review Questions

- How does an interface differ from an abstract class?
- Why does Java not support multiple inheritance? What feature of Java helps realize the benefits of multiple inheritance?
- An Abstract class must contain at least one abstract method, (true or false)?
- A subclass typically represents a larger number of objects than its super class, (true or false)?
- A subclass typically encapsulates less functionality than its super class does, (true or false)?
- An instance of a class can be assigned to a variable of type any of the interfaces the class implements, (true or false)?

Presentation Terminated