Introduction to Object Oriented Programming

MOHD HANAPI ABDUL LATIF
GLORIA JENNIS TAN

UITM TERENGGANU KAMPUS KUALA TERENGGANU

Chapter 1 Outline

Review of Object Oriented Programming

a) Object Oriented Programming Concepts

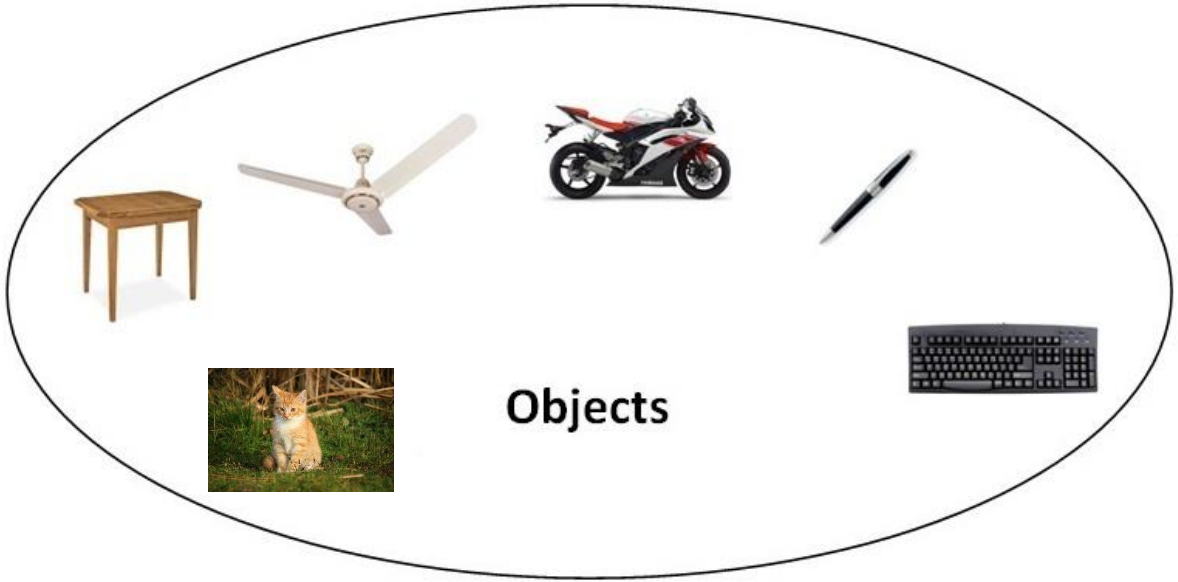
- Objects, classes, packages

b) Inheritance & Polymorphism concepts

- Inheriting instances fields and methods
- Method overriding
- Access levels –public, protected, private
- Abstract super classes and methods
- Interface

Objects, classes, packages

Introduction to object



Introduction to object

In broadest term, an object is a thing, both tangible, and intangible, that we can imagine.

Examples of real-world objects:

- cat, desk, television set, bicycle, etc.

Real world objects share two characteristics:

- They have **state** (represented by attributes and relationships)
- They have **behavior** (represented by operations and methods)

Object-oriented Programming

A new programming paradigm: think in term of objects.

- Objects have **attributes** and can do some **actions**.
- Objects can **interact** with one another.

Elements of an object: attribute, behavior, state

Cats

- Attribute: name, color
- Behavior: licking, jumping, running

Bicycle

- Attribute: speed, current gear
- Behavior: braking, changing gear

Software Objects

Modeled after real-world objects – have state and behavior

Maintains its state in variables.

Implements its behavior with methods/functions. Require all interaction to be performed through an object's method.

CLASSES

Class Definition

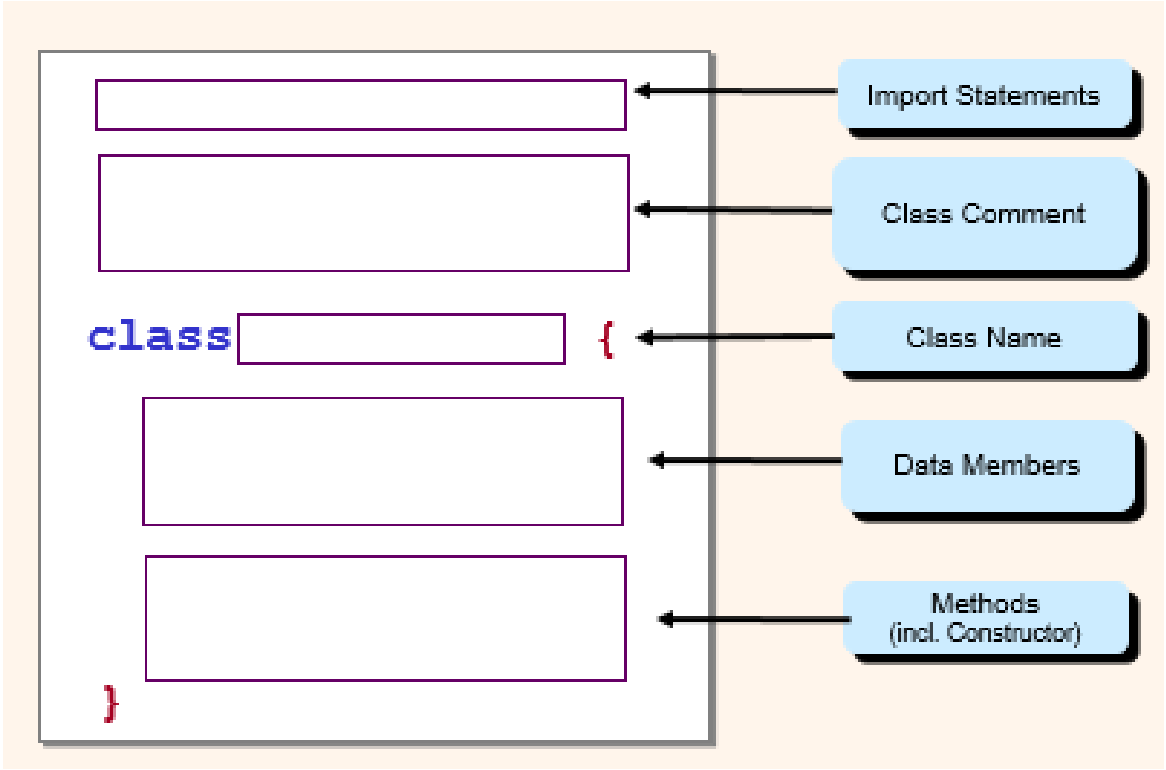
A class

- Template that describes what objects can and cannot do.
- The type for an object.
- A class is a blueprint that defines the data (attribute) and the methods (behavior) common to all objects of a certain kind.
- An object is called an *instance*of a class.
  - Eg. Class : Vehicle
  - Object: cars, MPVs, bicycles
- 2 types: user defined & pre-defined

Class Concept

- A class is a user defined blueprint or prototype from which objects are created.
- In general, class declarations can include these components, in order:
  1. Modifiers: A class can be public or has default access.
  2. Class name: The name should begin with a initial letter (capitalized by convention).
  3. Superclass (if any): The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
  4. Interfaces (if any): A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
  5.Body: The class body surrounded by braces, { }

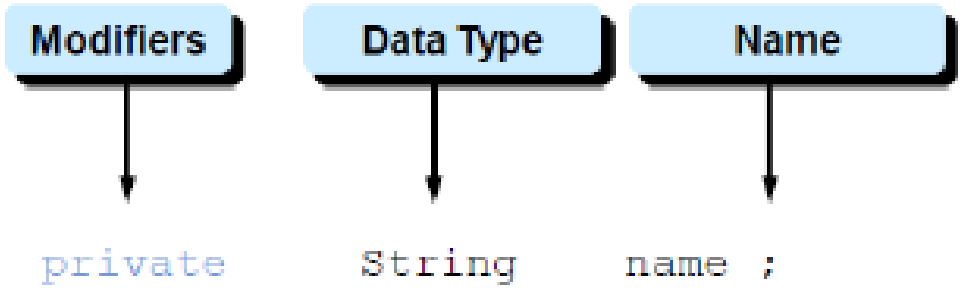
Class Definition



Class members

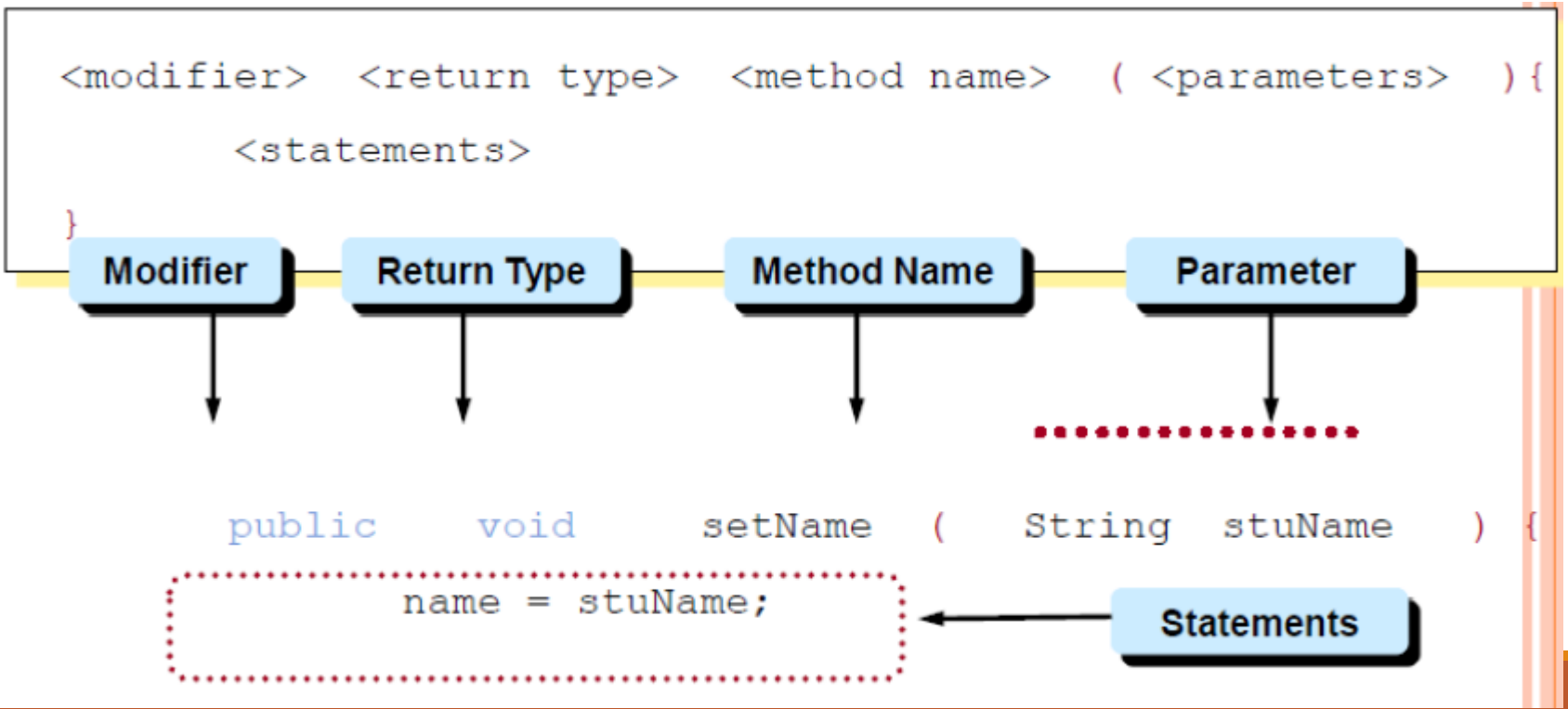
Data Member Declaration

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



Class members

Method Declaration



Constructor

- A *constructor* is a special method that is executed automatically when a new instance of the class (object) is created.
- Purpose: to initialize the data members of objects.
- Constructors characteristics:
  - has the **same name** as the class
  - It **executes automatically** when object of the class is created.
  - no return type hence no “return” statement

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

Constructor

Type of *constructor*

- Default constructor (no argument)

```
public    Student() {
     name = "Unassigned";
}
```

- Normal constructor (with argument)

```
public    Student(String name) {
     this.name = name;
}
```

An Accessor (Retriever)

Retrieve/read data members and return their values through the function type.

Usually labeled as *get*…e.g. *getData()*

Every class must have retriever method, possibly one for each data member.

```
public  String getName() {
    return name;
}
```

A Mutator (Storer)

- Store/write the data member based on the external values provided through the parameters.
- Usually labeled as *set*.. eg. *setData()*, *setName()*.

```
public void setName(String name) {

    this.name = name;

}
```

The Definition of the Student Class

```
public class Students {
     private String name; // Data Members
     private long id;
      // Default Constructor
     public Students() {
        name = "Unassigned";
        id = 0;
    }
     //Accessor: Returns the name of this student
     public String getName( ) {
        return name;
     }
     //Accessor: Returns the id of this student
     public long getId( ) {
        return id;
     }
    //Mutator: Assigns the name and id of this student
     public void setData(String name, long id) {
          this.name = name;
          this.id = id;
     }
     // display all information : return the string representation of the object
     public String toString() {
        return "Name: " + name +  "\n" + "Student id: " + id;
     }
}
```

```
/*    Author          : Anonymous
      Purpose         : This programs tests the class Students
*/

import java.util.*;
public class StuApp
{

      public static void main(String[] args)
      {
            Scanner in = new Scanner(System.in);

            Students std; // declare an object reference
            std   = new Students( ); // create an object of type Students

            System.out.print("Enter name: ");
            String nm = in.next();
            System.out.print("Enter id: ");
            long id = in.nextLong();

            std.setData(nm,id);

            System.out.println("The student's name: " + std.getName());
            System.out.println("The student's id: " + std.getId());
            System.out.println(std.toString());

      }// end main
}// end class
```

The Program Structure

StuApp.java

Student.java

To run the program: 1. javac Student.java (compile)
2. javac StuApp.java (compile)
3. java StuApp (run)

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

Information Hiding and Visibility Modifiers

The modifiers public and private designate the accessibility of data members and methods.

If a class component (data member or method) is declared private, client classes cannot access it.

If a class component is declared public, client classes can access it.

Internal details of a class are declared private and hidden from the clients. This is **information hiding**.

Accessibility Example

```
…
Service obj = new
Service();

obj.memberOne = 10;

obj.memberTwo = 20;

obj.doOne();

obj.doTwo();
…
```



```
class Service {
    public  int memberOne;
    private int memberTwo;
    public void doOne() {
      …
    }
    private void doTwo() {
      …
    }
}
```

Client

Service

Array of objects

- Array elements are not limited to primitive data-type but also and objects.
- Array also can be used to manipulate objects.
- Object can be duplicates more than one using object array.
- Some example.

Primitive Data-type	**Object**
`double[] rainfall;` `rainfall = new double[12];`	`Bicycle[] Bike;` `Bike = new Bicycle[10];`
`double[] rainfall = new double[12];`	`Bicycle[] Bike = new Bicycle[10];`

Arrays of Objects

- The use of an array of objects allows us to model the application more cleanly and logically.

```
public class Person {
    private String name;
    private int age;
    private char gender;

    public void setPerson(String name, int age, char gender)
    {
        this.name = name;
        this.age = age;
        this.gender = gender;
     }

     : // accessor
}
```

Arrays of Objects

We will use Person objects to illustrate the use of an array of objects.

```
Person std;

std = new Person();
std.setPerson("Aisyah", 20, 'F');

System.out.println("Name: " + std.getName());
System.out.println("Age : " + std.getAge());
System.out.println("Sex : " + std.getGender());
```

The Person class supports the set methods and get methods.

Creating an Object Array - 1

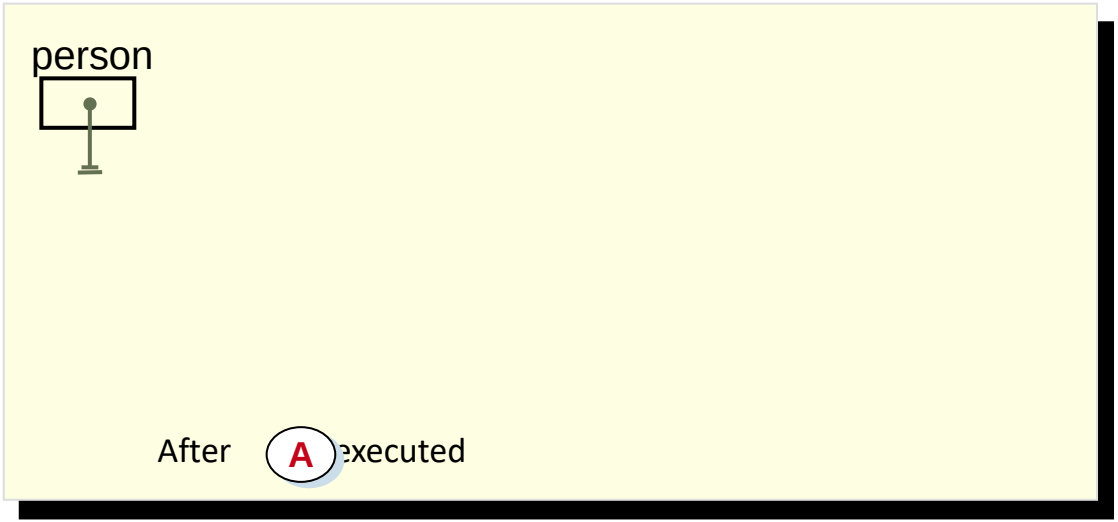
Code

A

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

Only the name person is declared, no array is allocated yet.

State of Memory



Creating an Object Array - 2

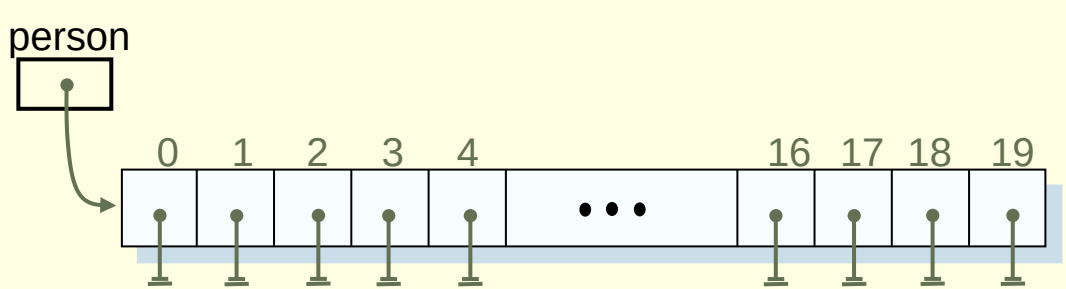
Code

B

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

Now the array for storing 20 Person objects is created, but the Person objects themselves are not yet created.

State of Memory



Creating an Object Array - 3

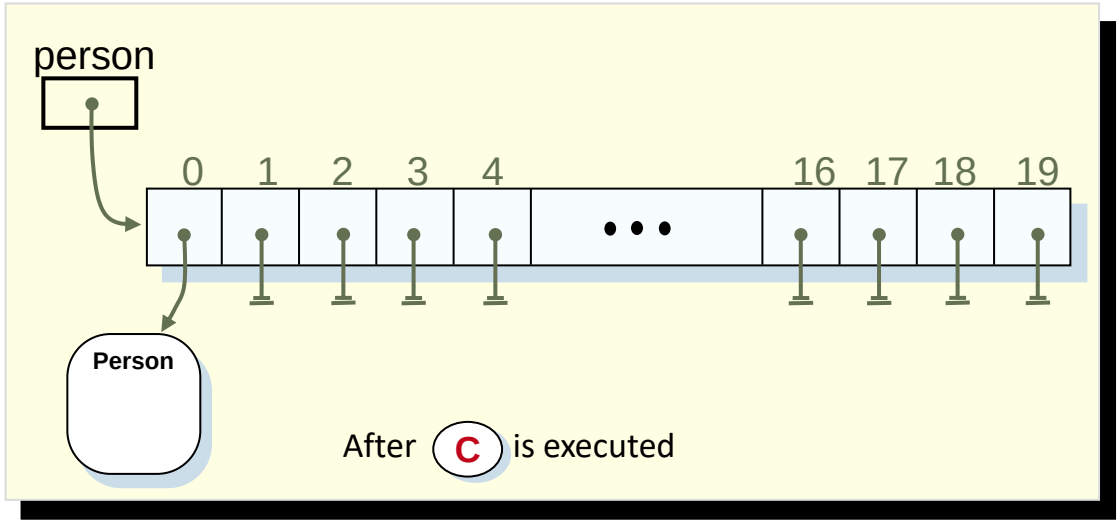
Code

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

C

One Person object is created and the reference to this object is placed in position 0.

State of Memory



Processing an Array of Objects with For-Each

```
Person[] person = new Person[100];
//create person[0] to person[99]
```

```
for (int i = 0; i < person.length; i++) {
    System.out.println(person[i].getName());
}
```

standard for loop

```
for (Person p : person) {
    System.out.println(p.getName());
}
```

for-each loop

For-Each: Key Points to Remember

A for-each loop supports read access only. The elements cannot be changed.

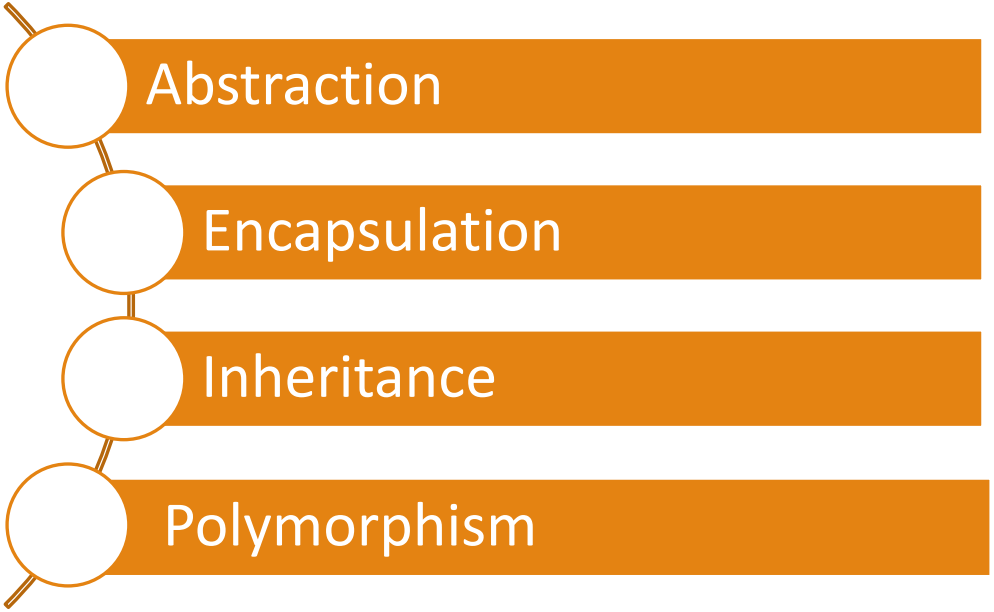
A single for-each loop allows access to a single array only, i.e., you cannot access multiple arrays with a single for-each loop.

A for-each loop iterates over every element of a collection from the first to the last element. You cannot skip elements or iterate backward.

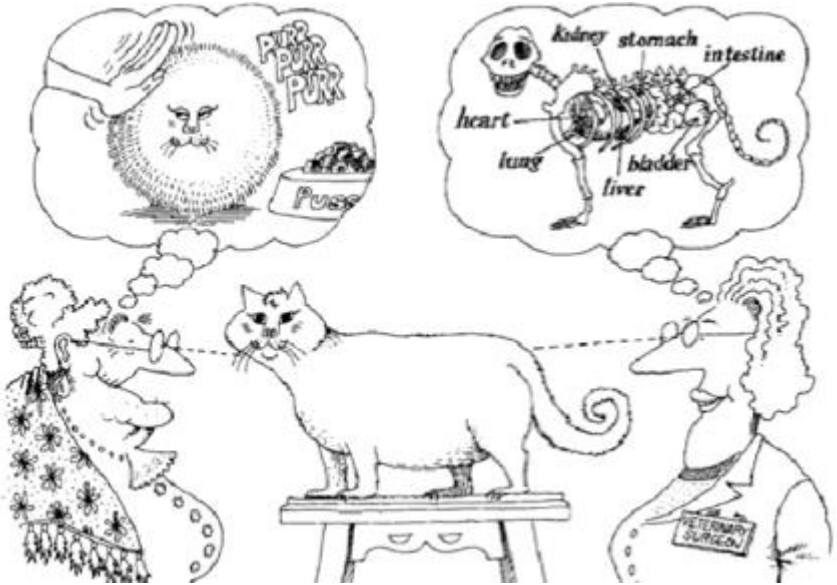
Characteristics of OOP

ABSTRACTION, ENCAPSULATION, INHERITANCE AND POLYMORPHISM

Characteristics of OOP

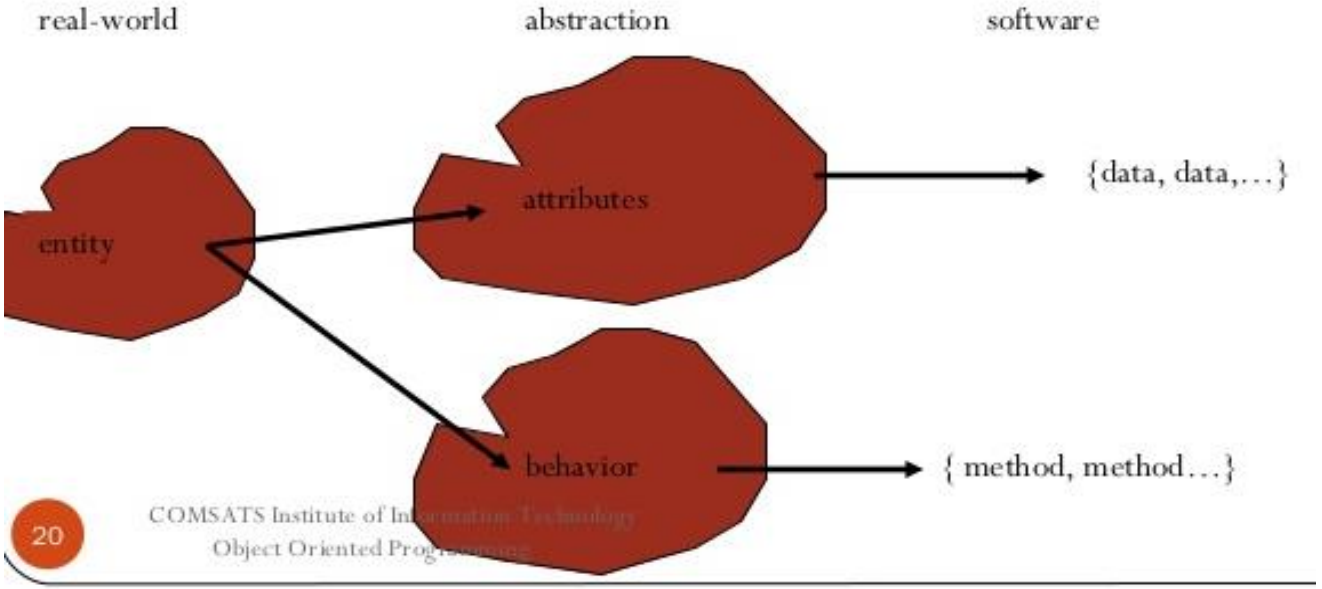


1 - Abstraction

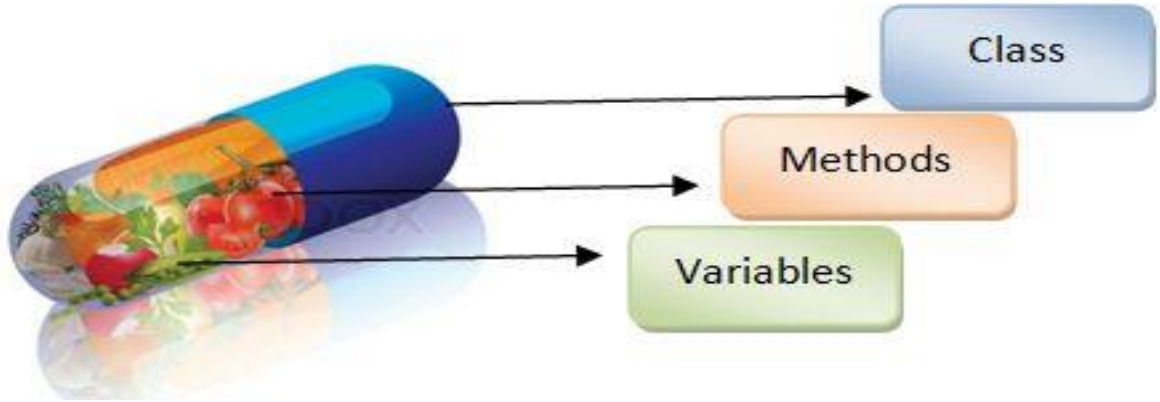


1 - Abstraction

Definition: Focuses upon the essential properties of an entity or object of interest. Depends on the perspective of viewers.



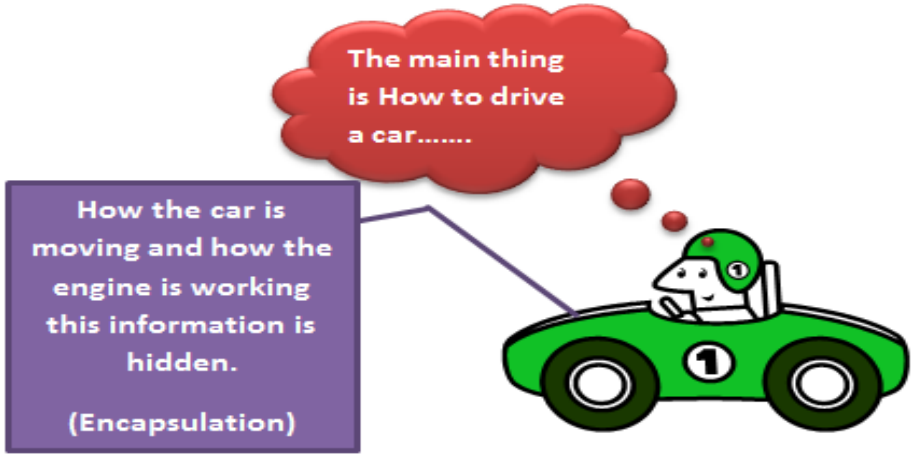
2 - Encapsulation



2 - Encapsulation

Definition: The process of hiding the variables and methods of an entity in a single unit (hides implementation from clients)

OOP encapsulates attributes and behaviors into packages called object.



3 - Inheritance

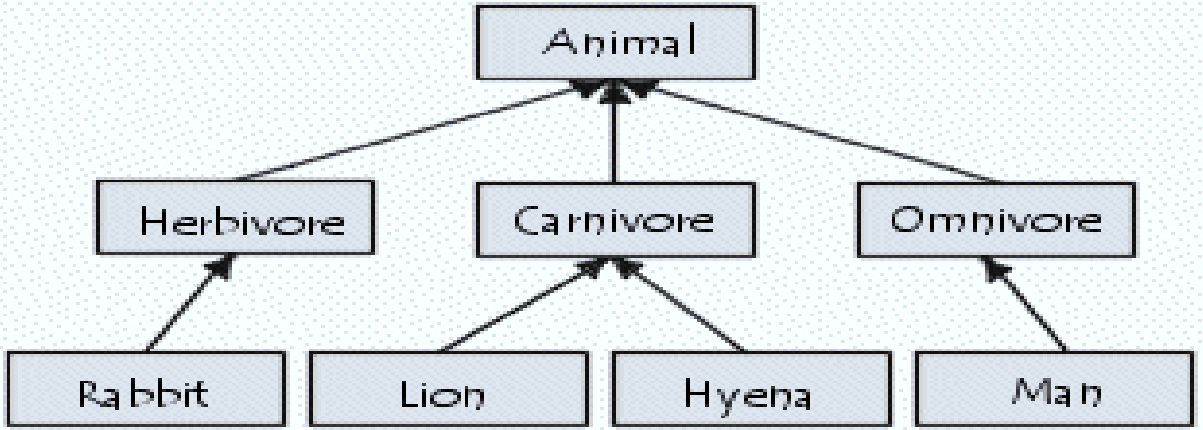
Inheritance is used to design two or more entities that are different, but share many common features.

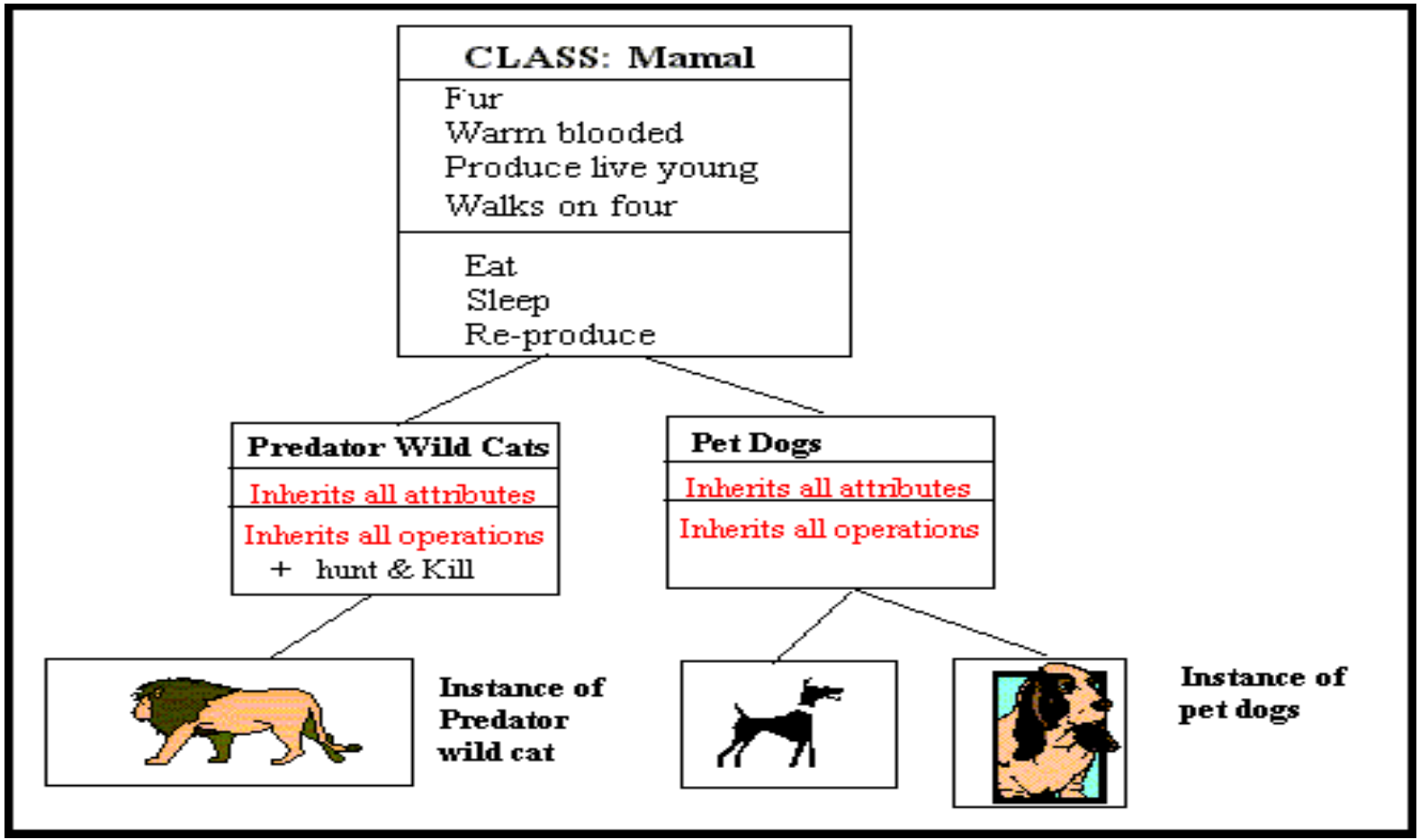
First, we define a class that contains the common features of the entities. Then we define classes as an extension of the common class, inheriting everything from common class.

We call the common class as the superclass, and all classes that inherit from it as subclasses.

Advantage: code/software reusability (safe time and cost)

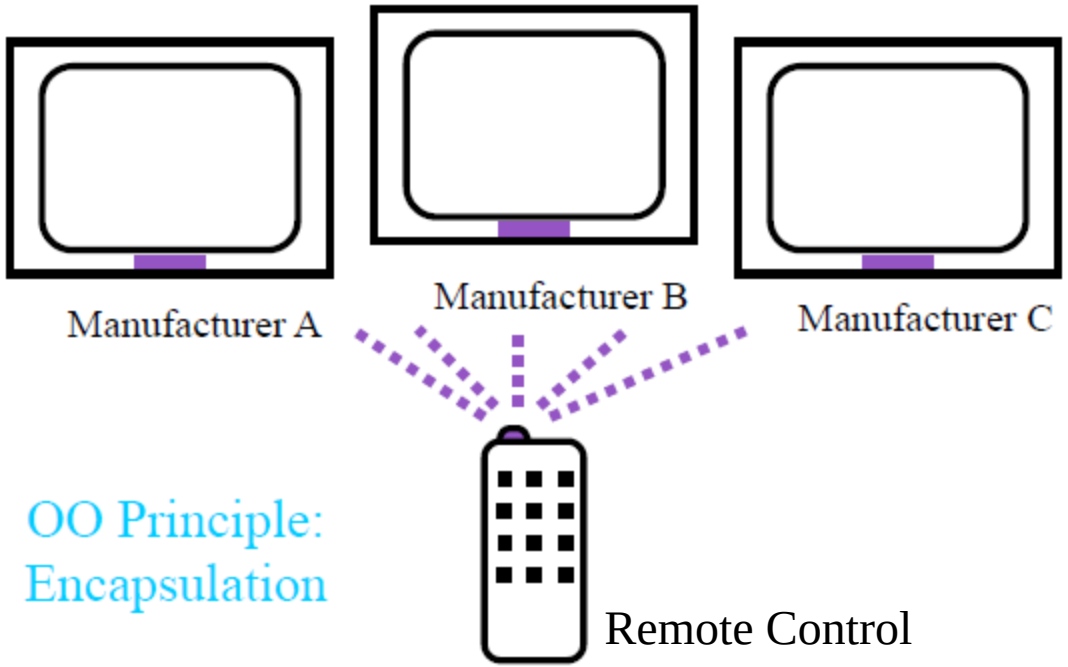
3 - Inheritance




CLASS: Mamal
Fur
Warm blooded
Produce live young
Walks on four
Eat
Sleep
Re-produce
Predator Wild Cats
Inherits all attributes
Inherits all operations
+ hunt & Kill
Pet Dogs
Inherits all attributes
Inherits all operations
Instance of Predator wild cat
Instance of pet dogs

4 - Polymorphism

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



4 - Polymorphism

Literally means “*many shapes*” or “*many forms*”.

Method overloading is one form of polymorphism: several methods with the same name may behave differently.

If a Dog is commanded to **speak()**, it may emit a bark, while if a Cat is asked to **speak()**, it may respond with a meow. Both inherit **speak()** from Animal, but their subclass methods override the methods of the superclass

4 - Polymorphism

- The ability of an object to take on many forms.
- The ability for objects of different classes related by inheritance to respond differently to the same method call.
- A **polymorphic** program is a program that allows a single variable to refer to objects from different classes.

Benefits of OOP

Bundling code into individual software objects provides a number of benefits, including:

- **Modularity**: The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.
- **Information-hiding**: By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
- **Code re-use**: If an object already exists (perhaps written by another software developer), you can use that object in your program.
- **Pluggability and debugging ease**: If a particular object turns out to be problematic, you can simply remove it from your application and plug in different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace it , not the entire machine.

Recall

What is object?

List two elements of an object.

List four basic concepts of OOP and their descriptions.

Activity 1

- What is the difference between classes and objects?
- Can polymorphism exist without inheritance?
- Given a problem to calculate the **area and perimeter** of a rectangle, explain how you can solve this problem using procedural approach. Then, try to solve the same problem using object-oriented programming approach.

Objects, classes, packages

PACKAGES

Package description

- Java allows you to **group classes** in a collection called a ***package***.
- Packages are convenient for organizing your work and for **separating your work from code libraries** provided by others.
- The standard Java library is distributed over a number of packages, including java.lang, java.util, java.net, and so on.
- All standard Java packages are inside the **java and javax package** hierarchies.
- The main reason for using packages is to **guarantee the uniqueness of class names**.

Activity 2

Write a simple java program that able to:

- Receive input from user (name and matrix id)
- Greet the user by displaying the name and matrix id

Lab Exercise

Write a JAVA program that has the following criteria

- 1 object with 3 attributes.
- Constructors (default, normal, setter, getter) 1 method that will do any related process for the object that you had specified. For example: method to display the object's details or method to calculate area of the shape.
- 1 toString() method to display all information