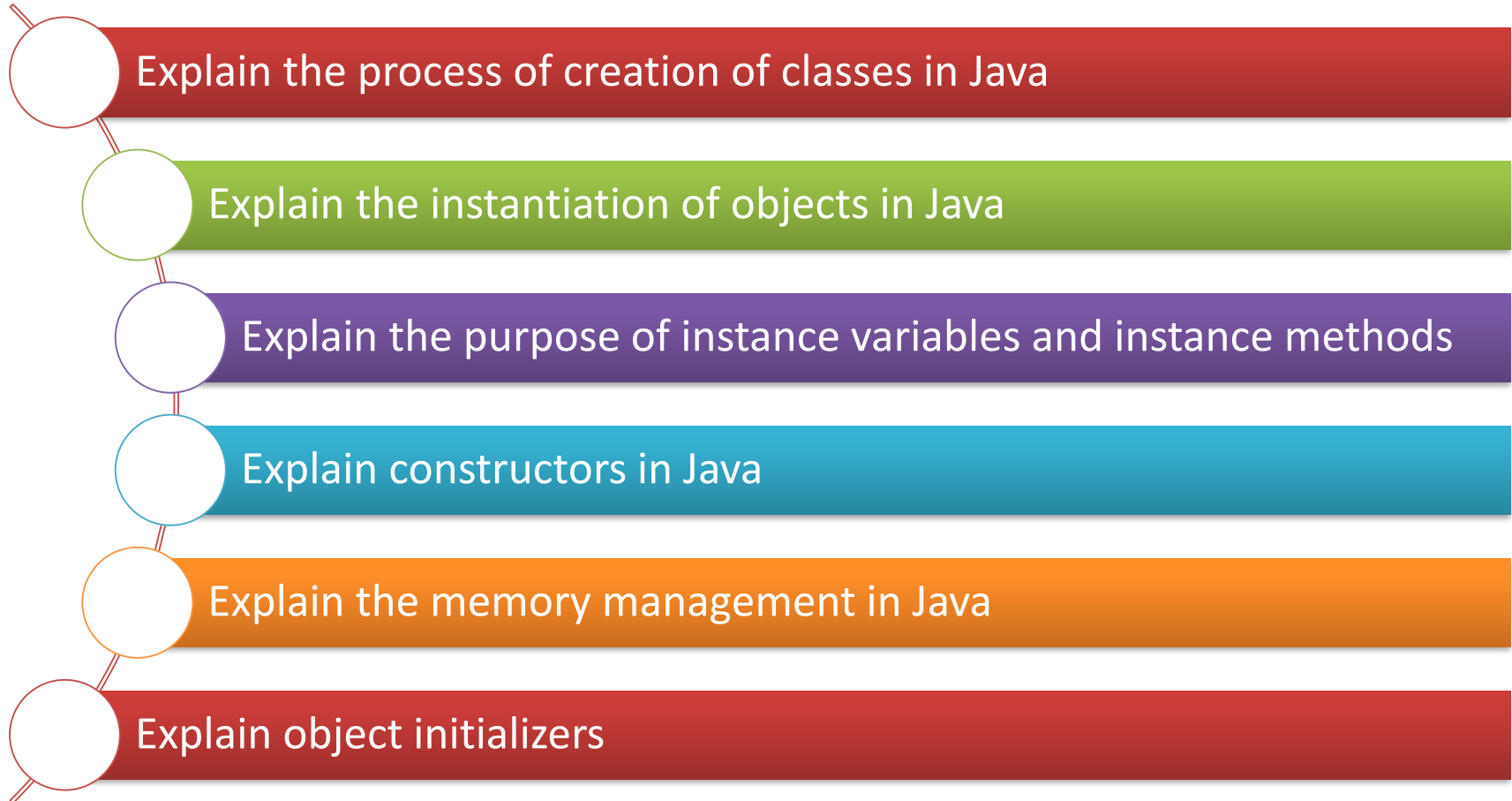




Session 03 Classes and Objects

Objectives

- 
- 1. Explain the process of creation of classes in Java
 - 2. Explain the instantiation of objects in Java
 - 3. Explain the purpose of instance variables and instance methods
 - 4. Explain constructors in Java
 - 5. Explain the memory management in Java
 - 6. Explain object initializers

■ Programming Languages:

- The development of software application is done using a programming language.
- A programming language is used as a medium for communicating the instruction to the computer.
- The programming language enforces a particular style of programming that is referred to as a programming paradigm.
- Following are the two types of programming paradigm:

Structured
Programming
Paradigm

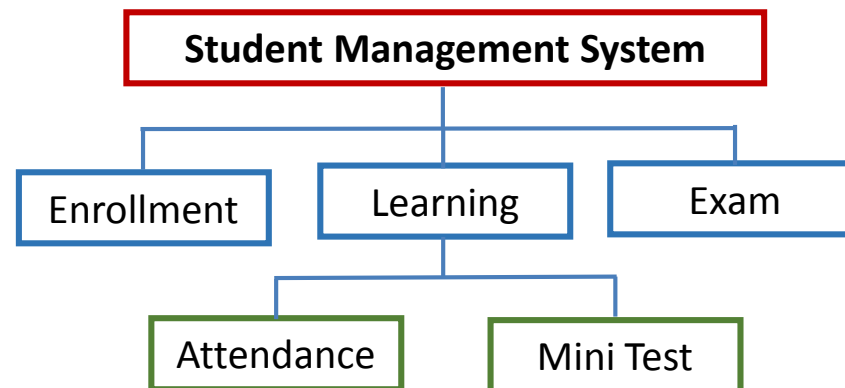
Object-oriented
Programming
Paradigm

Structured Programming Paradigm

Structured Programming

- In structured programming paradigm, the application development is decomposed into a hierarchy of subprograms.
- The subprograms are referred to as procedures, functions, or modules in different structured programming languages.
- Each subprogram is defined to perform a specific task.
- Some of structured programming languages are C, Pascal, and Cobol.

- Following figure displays school application activities broken down into subprograms:



Code example: Enrollment

```

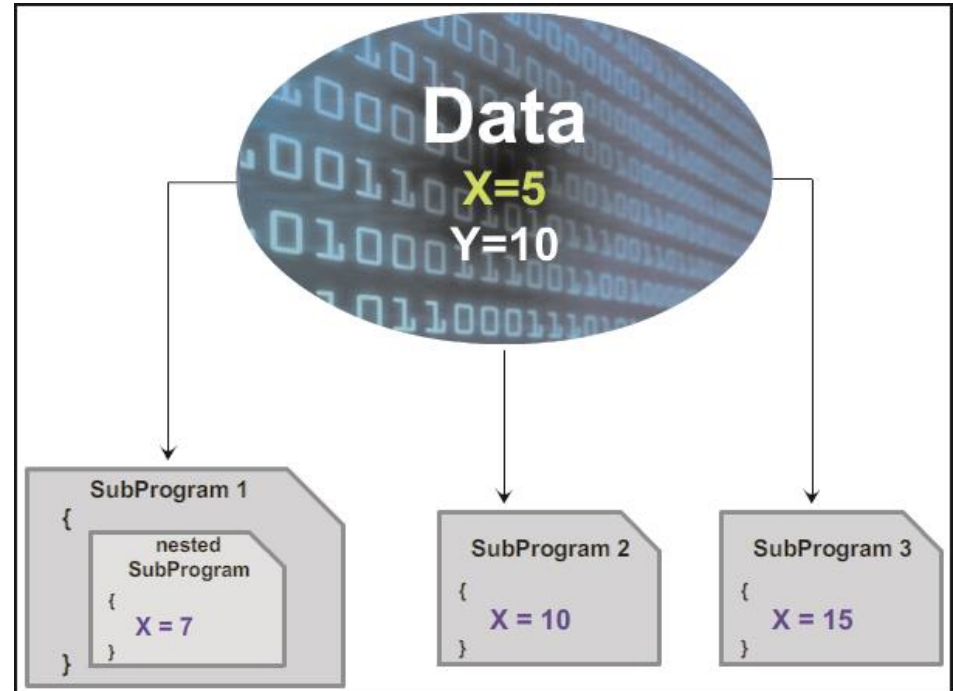
struct Student{
    char name[30];
    //.....
};
void enroll(Student &sv){
    printf("\nName: ");
    fflush(stdin); gets(sv.name);
    //.....
}
void print(Student sv){
    printf("\nName: %s", sv.name);
    //.....
}
int main(){
    Student a;
    enroll(a);
    print(a);
}

```

Structured Programming Paradigm

■ Main disadvantage of structured programming languages are as follows:

- Data is shared globally between the subprograms.
- Efforts are spent on accomplishing the solution rather than focusing on problem domain.



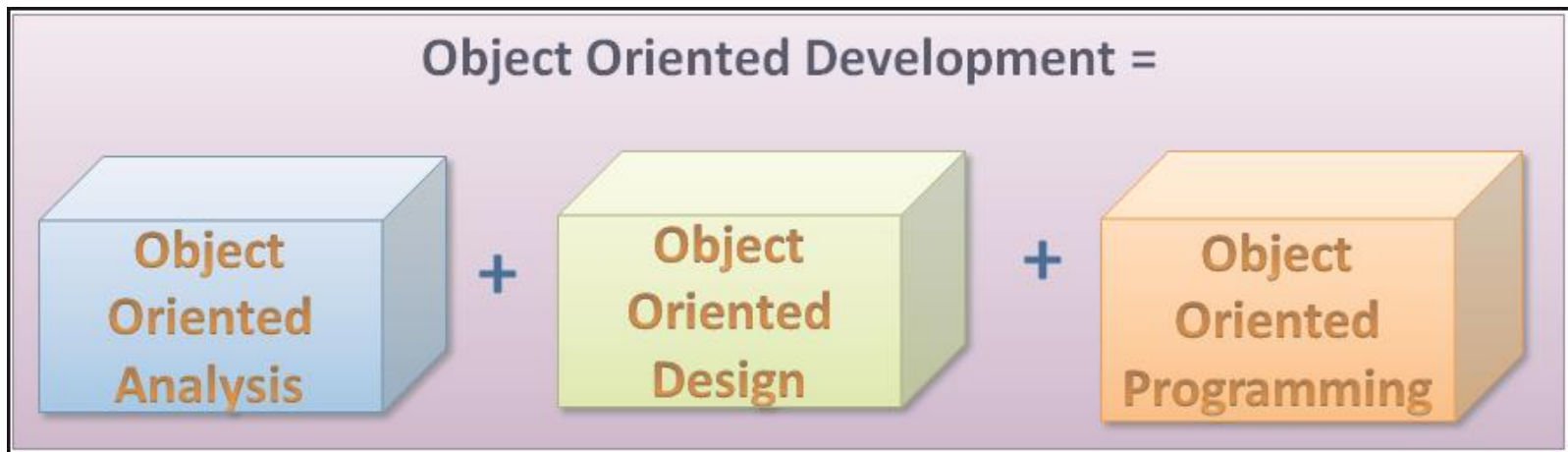
- ◆ This often led to a software crisis, as the maintenance cost of complex applications became high and availability of reliable software was reduced.

Object-oriented Programming Paradigm

- Growing complexity of software required change in programming style.
- Some of the features that were aimed are as follows:
 - Development of reliable software at reduced cost.
 - Reduction in the maintenance cost.
 - Development of reusable software components.
 - Completion of software development with the specified time interval.
- These features resulted in the evolution of **object-oriented programming paradigm.**

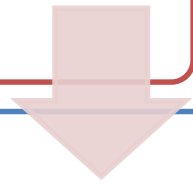
Object-oriented Programming Paradigm

- The software applications developed using object-oriented programming paradigm is:
 - Designed around data, rather than focusing only on the functionalities.
- Following shows different activities involved in the object-oriented software development:



Object-oriented Programming Paradigm

Object-oriented Analysis (OOA) phase determines the functionality of the system.



Object-oriented Design (OOD) phases determines the process of planning a system in which objects interact with each other to solve a software problem.



Object-oriented Programming (OOP) deals with the actual implementation of the application.

- Unified Modeling Language (UML) helps to create visual models in the system.
- The actual implementation of these visual models is done using an OOP language.

Principles of OOP



Object – Represents an entity which possesses certain features and behaviors.

Class – Is a template that is used to create objects of that class.

Abstraction – Is a design technique that focuses only on the essential features of an entity for a specific problem domain.

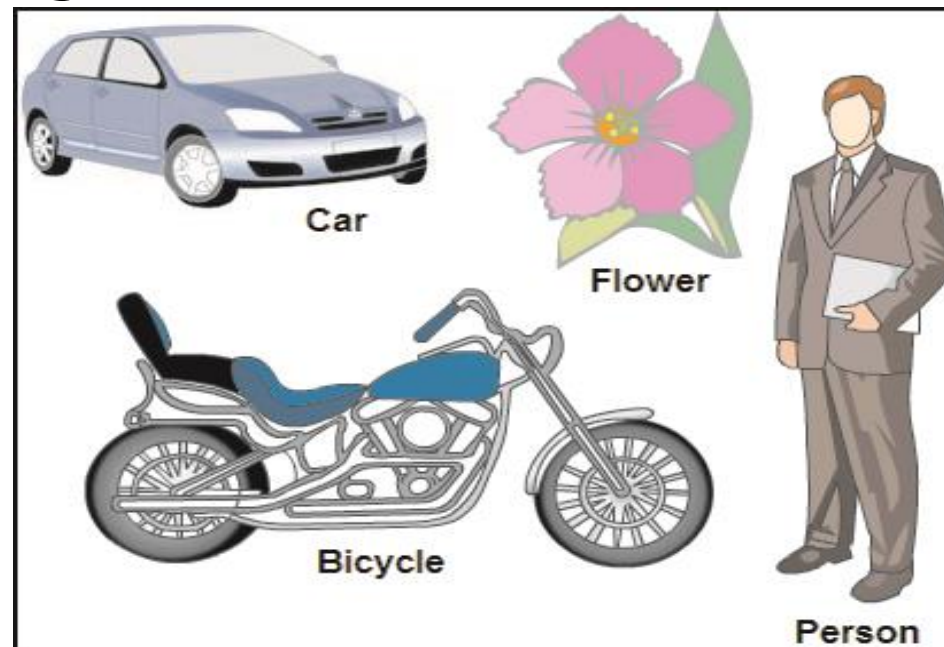
Encapsulation – Is a mechanism that combines data and implementation details into a single unit called class.

Inheritance – Enables the developer to extend and reuse the features of existing classes and create new classes. The new classes are referred to as derived classes.

Polymorphism – Is the ability of an object to respond to same message in different ways.

Concept of an Object

- An object represents a real-world entity.
- Any tangible or touchable entity in the real-world can be described as an object.
- Following figure shows some real-world entities:



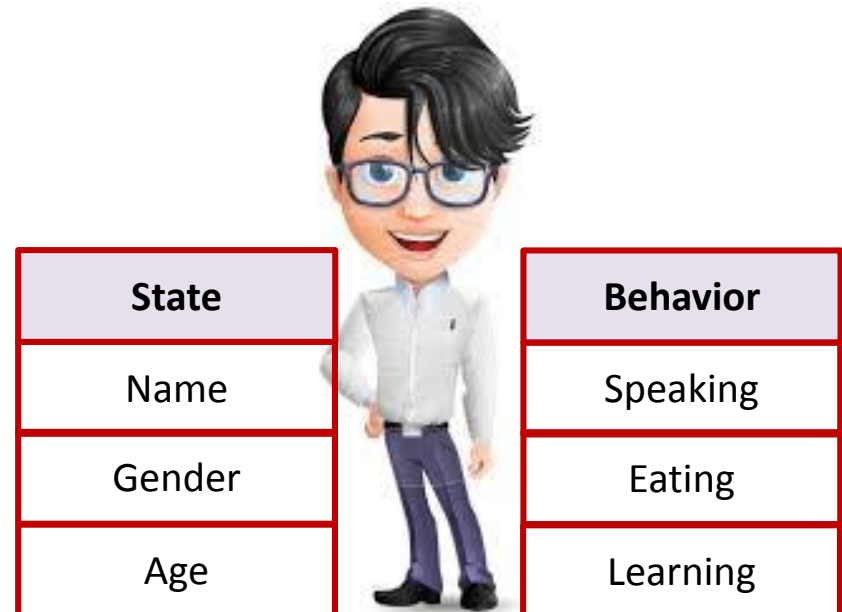
Concept of an Object

- Each object has:
 - **Characteristics** – Defined as attributes, properties, or features describing the object.
 - **Actions** – Defined as activities or operations performed by the object.
- ◆ Example of an object, **Person**.
 - ◆ **Properties** – Name, Gender, Age, ...
 - ◆ **Actions** – Speaking, Eating, Learning, ...
- ◆ The concept of objects in the real-world can be extended to the programming world where software 'objects' can be defined.



Concept of an Object

- A software object has state and behavior.
- '**State**' refers to object's characteristics or attributes.
- '**Behavior**' of the software object comprises its actions.
- Following figure shows a software object, a **Person** with its state and behavior:



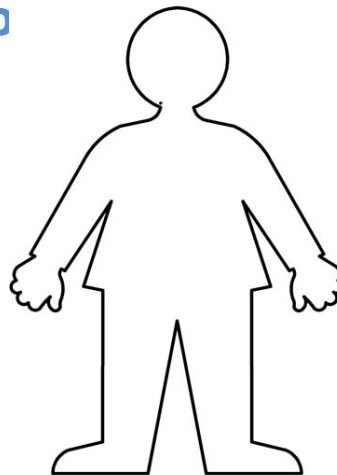
Concept of an Object

The advantages of using objects:

- They help to understand the real-world.
- They map attributes and actions of real-world entities with state and behavior of software objects.

Defining a Class

- In the real-world, several objects:
 - Have common state and behavior.
 - Can be grouped under a single class.
 - Example: All person objects have attributes, such as color, make, or model.
- **Class:**
 - Can be defined that a class is a template or blueprint which defines the state and behavior for all objects belonging to that class.
 - Following figure shows **a person as a template** and a **John as an object** or **instance o**



Class

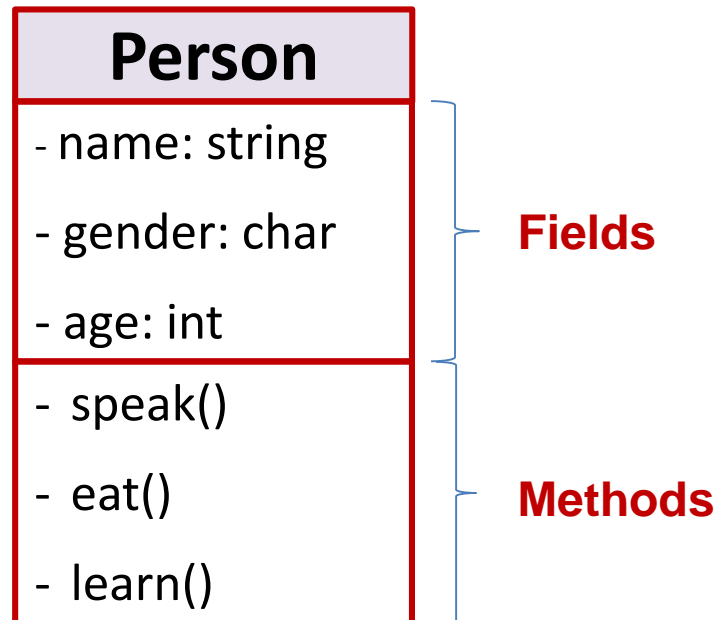


Object

State
Name: John
Gender: Male
Age: 20

Defining a Class

- Class comprises fields and methods, collectively called as members.
 - **Fields** – Are variables that depict the state of objects.
 - **Methods** – Are functions that depict the behavior of objects.



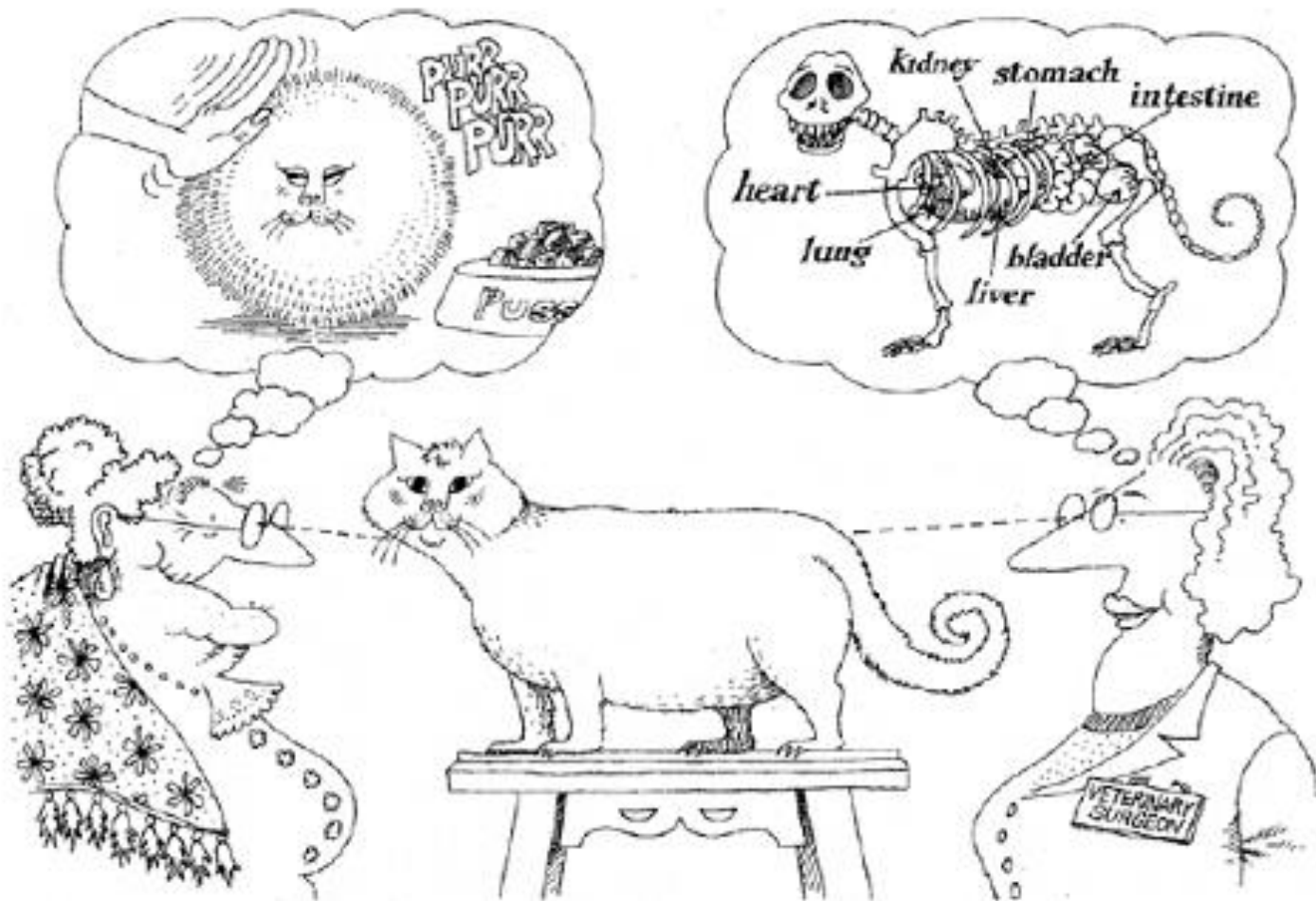
Defining a Class

- Following table shows the difference between a class and an object:

Class	Object
Class is a conceptual model	Object is a real thing
Class describes an entity	Object is the actual entity
Class consists of fields (data members) and functions	Object is an instance of a class

1. Abstraction

■ Abstraction **focuses** upon the **essential characteristics** of some object, relative to the perspective of the viewer



1. Abstraction

Employee

- name: string
- year_of_birth: int
- time:: int
- Salary: int
- recruit()
- timeTrack()
- payroll()

OFFICE



Student

- name: string
- gender: char
- year_of_birth: int
- GPA: float
- enroll()
- exam()
- learn()

SCHOOL

Declaring a class

- The syntax to declare a class in Java is as follows:

Syntax

```
class <class_name> {  
    // class body  
}
```

- The body of the class is enclosed between the curly braces { }.
- In the class body, you can declare members, such as fields, methods, and constructors.

Declaring a class

```
class Student {
    String name;
    //.....
    float GPA;

    void initialize(){
        name = "no name";
        //.....
        GPA = 0;
    }

    void enroll(){
        Scanner scanner = new Scanner(System.in);
        System.out.println("Name: ");
        //.....
    };

    void exam(float GPA){
        this.GPA = GPA;
    }

    void print(){
        System.out.println("Name: "+name);
        //.....
        System.out.println("GPA: "+GPA);
    };
}
```

Fields or
Instance Variables

Functions or
Instance Methods

Creating an Object

- An object is created using the `new` operator.
- On encountering the `new` operator:
 - JVM allocates memory for the object.
 - Returns a reference or memory address of the allocated object.
 - The reference or memory address is then stored in a variable called as reference variable.
- The syntax for creating an object is as follows:

Syntax

```
<class_name> <object_name> = new <class_name> ();
```

where,

`new`: Is an operator that allocates the memory for an object at runtime.

`object_name`: Is the variable that stores the reference of the object.

Creating an Object

- Following code snippet demonstrates the creation of an object in a Java program:

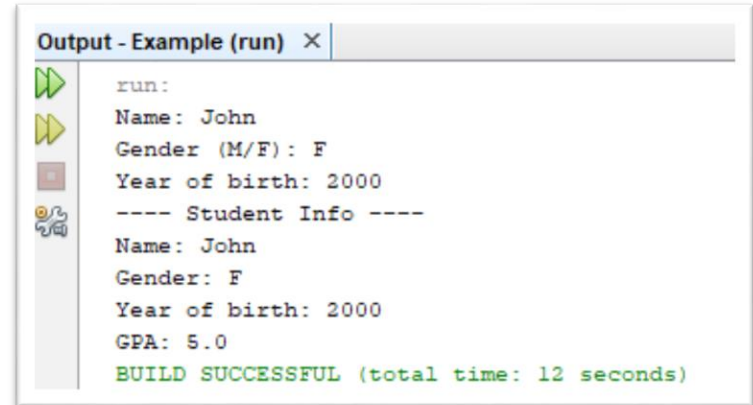
```
Student objStudent = new Student ();
```

- The expression on the right side, **new Student()** allocates the memory at runtime.
- After the memory is allocated for the object, it returns the reference or address of the allocated object, which is stored in the variable, **objStudent**

Creating an Object

```
public class Example {

    public static void main(String[] args) {
        Student objStudent = new Student();
        objStudent.enroll();
        objStudent.exam(5);
        objStudent.print();
    }
}
```



```
Output - Example (run) X
run:
Name: John
Gender (M/F): F
Year of birth: 2000
---- Student Info ----
Name: John
Gender: F
Year of birth: 2000
GPA: 5.0
BUILD SUCCESSFUL (total time: 12 seconds)
```

2. Encapsulation

Structured Pro.. VS OOP

```
class Student {
    String name;
    //.....
    float GPA;
    void initialize(){
        name = "no name";
        //.....
        GPA = 0;
    }
    void enroll(){
        Scanner scanner = new Scanner(System.in);
        System.out.println("Name: ");
        //.....
    };
    void exam(float GPA){
        this.GPA = GPA;
    }
    void print(){
        System.out.println("Name: "+name);
        //.....
        System.out.println("GPA: "+GPA);
    };
}

public class Example {

    public static void main(String[] args) {
        Student objStudent = new Student();
        objStudent.enroll();
        objStudent.exam(5);
        objStudent.print();
    }
}
```

```
struct Student{
    char name[30];
    //.....
};

void enroll(Student &sv){
    printf("\nName: ");
    fflush(stdin); gets(sv.name);
    //.....
}

void print(Student sv){
    printf("\nName: %s", sv.name);
    //.....
}

int main(){
    Student a;
    enroll(a);
    print(a);
}
```

Creation of an Object: Two Stage Process

- Alternatively, an object can be created using two steps that are as follows:
 - Declaration of an object reference.
 - Dynamic memory allocation of an object.
- **Declaration of an object reference:**
 - The syntax for declaring the object reference is as follows:

Syntax

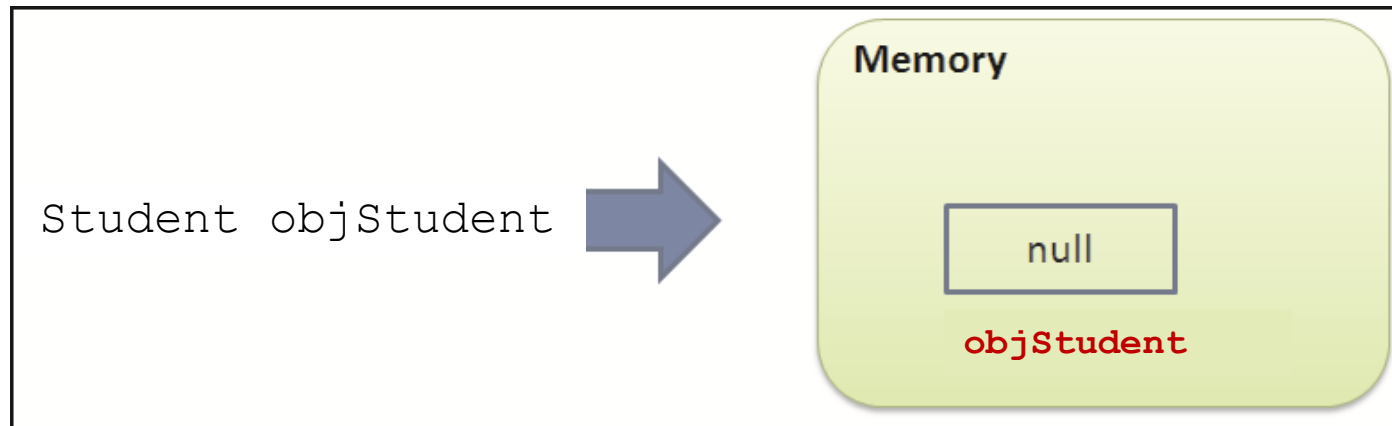
```
<class_name> <object_name>;
```

where,

`object_name`: Is just a variable that will not point to any memory location.

Creation of an Object: Two Stage Process

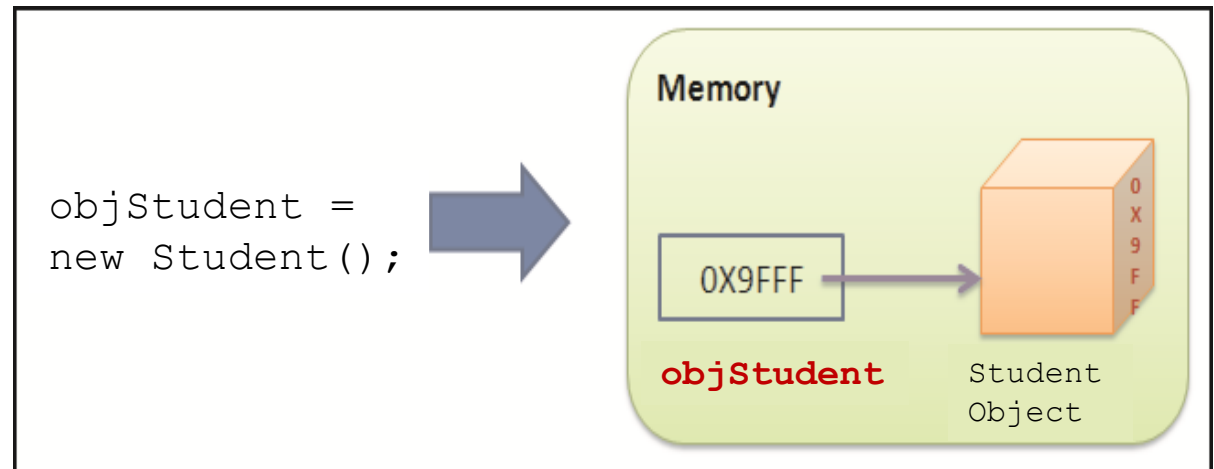
- Following figure shows the effect of the statement, `Student objStudent;` which declares a reference variable:



- By default, the value `null` is stored in the object's reference variable which means reference variable does not point to an actual object.
- If `objStudent` is used at this point of time, without being instantiated, then the program will result in a compile time error.

Creation of an Object: Two Stage Process

- **Dynamic memory allocation of an object:**
 - The object should be initialized using the `new` operator which dynamically allocates memory for an object.
 - For example, the statement, `objStudent = new Student()`; allocates memory for the object and memory address of the allocated object is stored in the variable `objStudent`.
- Following figure shows the creation of object in the memory and storing of its reference in the variable, `objStudent`:



Members of a Class

- The members of a class are fields and methods.

Fields

- Define the state of an object created from the class.
- Referred to as instance variables.

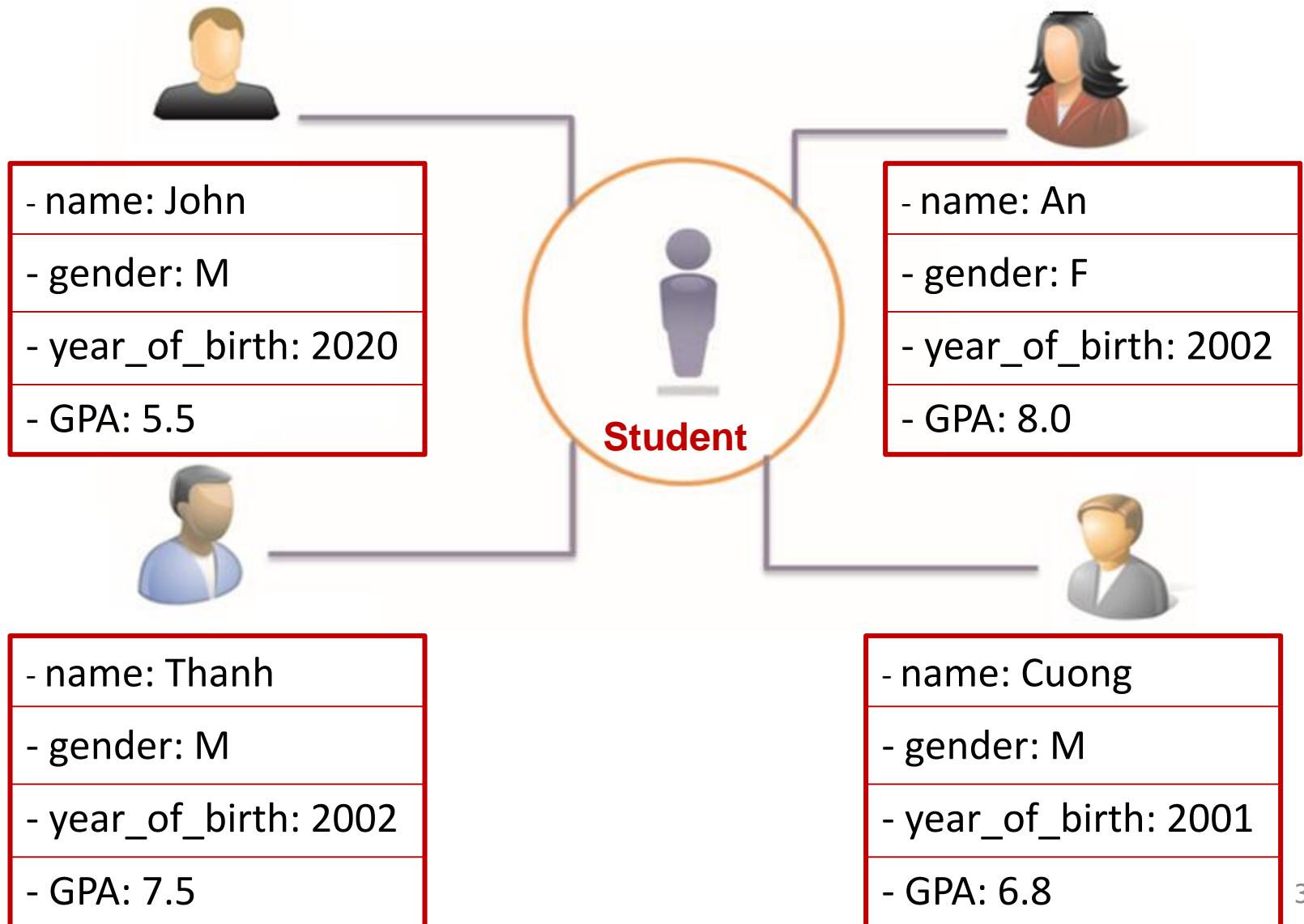
Methods

- Implement the behavior of the objects.
- Referred to as instance methods.

Instance Variables

- They are used to store data in them.
- They are called instance variables because each instance of the class, that is, object of that class will have its own copy of the **instance variables**.
- They are declared similar to local variables.
- They are declared inside a class, but outside any method definitions.
- For example: Consider a scenario where the **Student** class represents the details of students in a school.
 - A typical question that can be asked is 'What are the different data that are required to identify a student in a school domain and represent it as a single object?'

Instance Variables



Instance Variables

- The syntax to declare an instance variable within a class is as follows:

Syntax

```
[access_modifier] data_type instanceVariableName;
```

where,

`access_modifier`: Is an optional keyword specifying the access level of an instance variable. It could be `private`, `protected`, and `public`.

`data_type`: Specifies the data type of the variable.

`instanceVariableName`: Specifies the name of the variable.

- Instance variables are accessed by objects using the dot operator (`.`).

Instance Methods

- They are functions declared in a class.
- They implement the behavior of an object.
- They are used to perform operations on the instance variables.
- They can be accessed by instantiating an object of the class in which it is defined and then, invoking the method.
- For example: the class `Student` can have a method `Enroll()` that represents the 'Input Student Information' action.
 - To perform the action, the method `Enroll()` will have to be invoked by an object of class `Student`.

Conventions to be followed while naming a method are as follows:

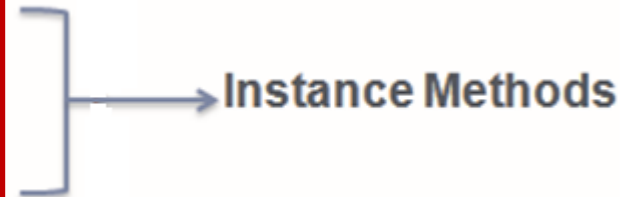
- Cannot be a Java keyword.
- Cannot contain spaces.
- Cannot begin with a digit.
- Can begin with a letter, underscore, or a '\$' symbol.
- Should be a verb in lowercase.
- Should be descriptive and meaningful.
- Should be a multi-word name that begins with a verb in lowercase, followed by adjectives, nouns, and so forth.

Instance Methods

Student

- name: string
- gender: char
- year_of_birth: int
- GPA: float

- enroll()
- exam()
- learn()



Instance Methods

- The syntax to declare an instance method in a class is as follows:

Syntax

```
[access_modifier] <return_type> <method_name> ([list  
of parameters]) {  
  
// Body of the method  
  
}
```

where,

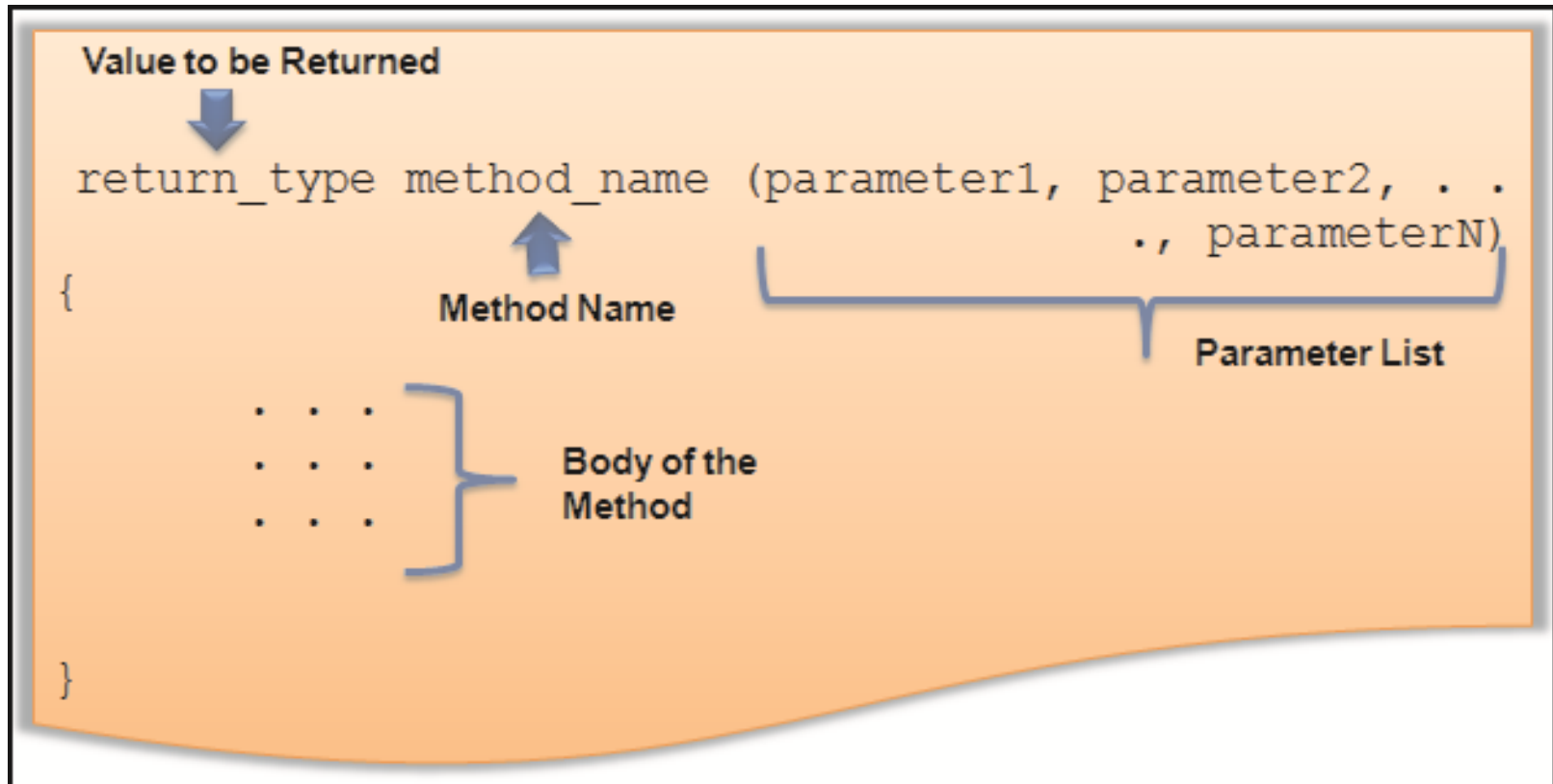
access_modifier: Is an optional keyword specifying the access level of an instance method. It could be `private`, `protected`, and `public`.

returntype: Specifies the data type of the value that is returned by the method.

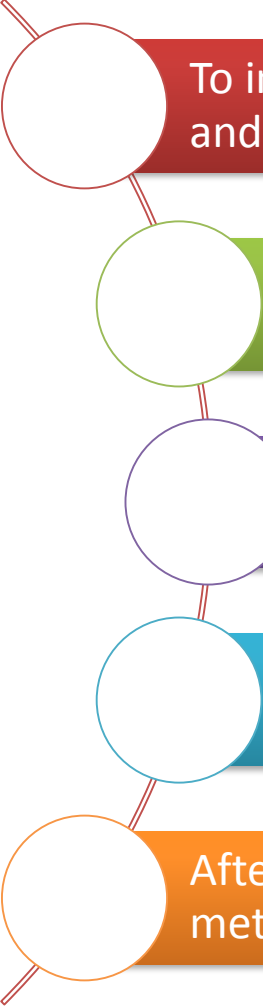
method_name: Is the method name.

list of parameters: Are the values passed to the method.

Instance Methods



Invoking Methods



To invoke a method, the object name is followed by the dot operator (.) and the method name.

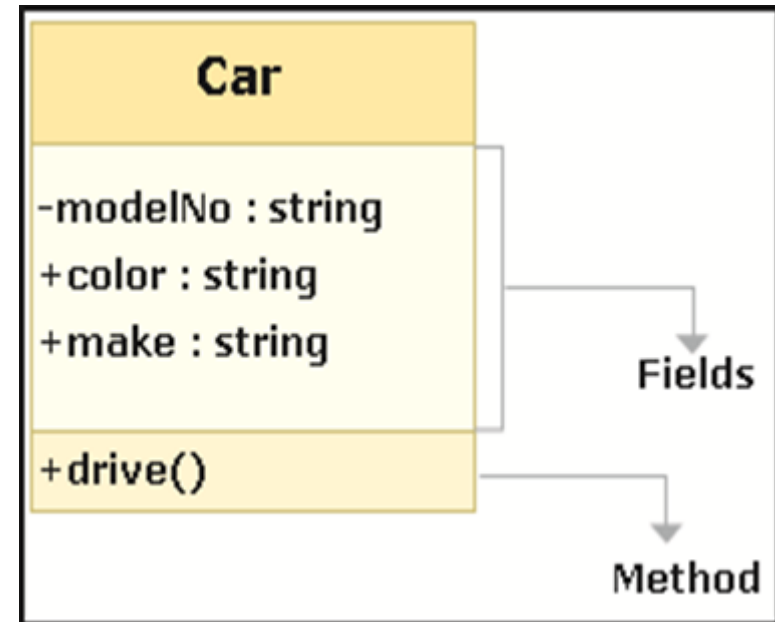
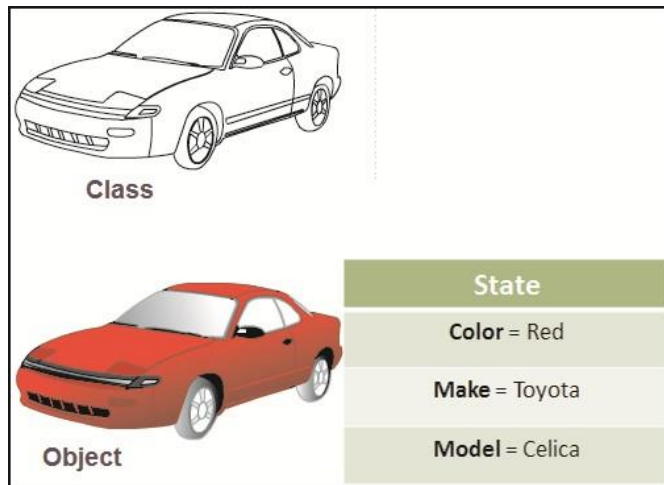
A method is always invoked from another method.

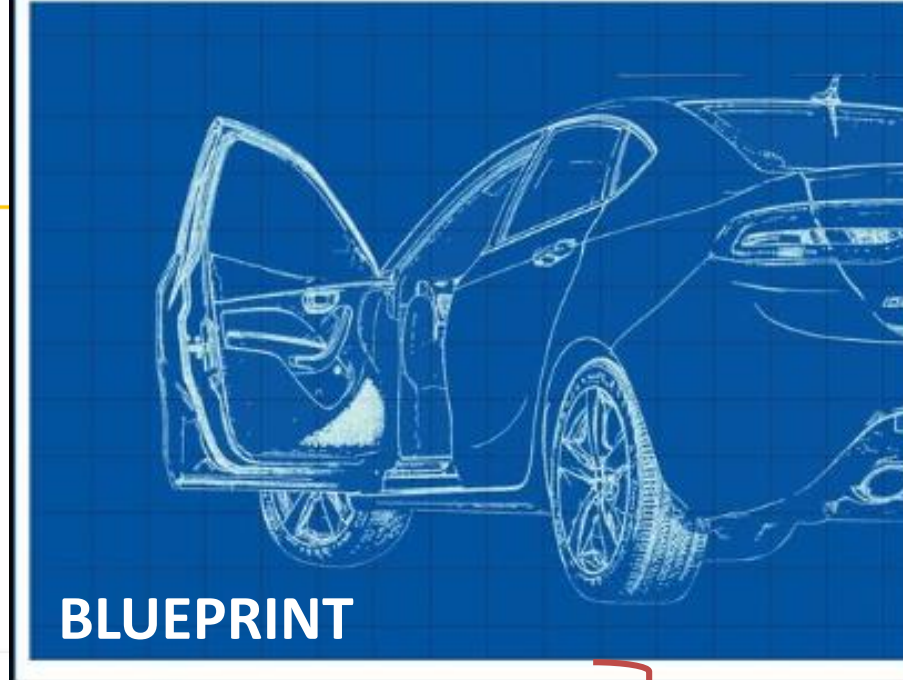
The method which invokes a method is referred to as the **calling method**.

The invoked method is referred to as the **called method**.

After execution of all the statements within the code block of the invoked method, the control returns back to the **calling method**.

Example: Car





```
public class Car {  
    private int yearModel;  
    private String make;  
    private int speed;  
    public Car (int yrModel, String carMake)  
    public void setyearModel(int yrModel)  
    public void getMake (String carMake)
```

CLASS

{...5 lines }

{...3 lines }

{...2 lines }

```
public class HelloWorld {
```

```
    public static void main(String[] args) {  
        Car carl = new Car(89 , "Tempo");  
        System.out.println("Year Model of carl: " + carl.getYearModel());  
    }
```

OBJECT

```

public class Car {
    private int yearModel;
    private String make;
    private int speed;
    public Car (int yrModel, String carMake) { ...5 lines }
    public void setyearModel(int yrModel) {
        yearModel = yrModel;
    }
    public void setMake (String carMake) {
        make = carMake;
    }
    public void setSpeed(int carSpeed) { ...3 lines }
    public int getYearModel() {
        return yearModel;
    }
    public String getMake () { ...3 lines }
    public int getSpeed () { ...3 lines }
    public void AccelerateSpeed (int speed) { ...3 lines }
    public void BrakeSpeed (int speed) { ...3 lines }
}

```

Instance Variables

Instance Methods

Student Activities

Employee

- name: string
- year_of_birth: int
- time:: int
- salary: int
- recruit()
- timeTrack()
- payroll()
- print()

■ Instance method

- recruit(): permit user input name, year of birth and salary (money/hour) of employee
- timeTrack(): time++
- payroll(): print time*salary and set time =0;
- print(): print employee info

■ Main:

- permit user input information of an employee
- Permit user input n employees and print their info (salary desc)

Student Activities

Coordinate

- x: int
- y: int
- setXY(int, int)
- entry()
- setX(int)
- setY(int)
- getX(int)
- getY(int)
- move(int,int)

■ Instance method

- set...(): set its value = parameter
- get...(): return its value
- entry(): permit user entry value
- move(int, int): move this point to dx, dy ($x += dx$ $y += dy$)

■ Main

- Declare 2 point of line segment
- Permit user entry value
- Print them



Slot06 cont...

- It is a method having the same name as that of the class.
- It initializes the variables of a class or perform startup operations only once when the object of the class is instantiated.
- It is automatically executed whenever an instance of a class is created.
- It can accepts parameters and do not have return types.
- It is of two types:
 - No-argument constructor
 - Parameterized constructor
- Following figure shows the constructor declaration:

```
class <ClassName>
{
    <ClassName> () ← Constructor
    {
        // Initialization code
    }
}
```

Invoking Constructor

- The constructor is invoked immediately during the object creation which means:
 - Once the `new` operator is encountered, memory is allocated for the object.
 - Constructor method is invoked by the JVM to initialize the object.
- Following figure shows the use of `new` operator to understand the constructor invocation:

```
<class_name> <object_name> = new <class_name>();
```

The parenthesis after the class name indicates the invocation of the constructor.

Default Constructor

- Consider a situation, where the constructor method is not defined for a class.
- In such a scenario, an implicit constructor is invoked by the JVM for initializing the objects.
- This implicit constructor is also known as default constructor.
- **Default Constructor:**
 - Created for the classes where explicit constructors are not defined.
 - Initializes the instance variables of the newly created object to their default values.

Parameterized Constructor



The parameterized constructor contains a list of parameters that initializes instance variables of an object.

The value for the parameters is passed during the object creation.

This means each object will be initialized with different set of values.

Parameterized Constructor

```
class Student {
    String name;
    char gender;
    int year_of_birth;
    float GPA;
    Student() {
        name = "fpr student";
        gender = 'M';
        year_of_birth = 2000;
        GPA = 8;
    }
    Student(String name, char gender, int year_of_birth, float GPA) {
        this.name = name;
        this.gender = gender;
        this.year_of_birth = year_of_birth;
        this.GPA = GPA;
    }
    //.....
    void print() {
        System.out.println("---- Student Info ---- ");
        System.out.println("Name: "+name);
        System.out.println("Gender: "+gender);
        System.out.println("Year of birth: "+year_of_birth);
        System.out.println("GPA: "+GPA);
    }
}
```

```
public class Example {
    public static void main(String[] args) {
        Student objStudent1 = new Student();
        objStudent1.print();
        Student objStudent2 = new Student("Anh", 'M', 2000, 8);
        objStudent2.print();
    }
}
```

Output - Example (run) X

```
run:
---- Student Info ----
Name: fpr student
Gender: M
Year of birth: 2000
GPA: 8.0
---- Student Info ----
Name: Anh
Gender: M
Year of birth: 2000
GPA: 8.0
BUILD SUCCESSFUL (total time: 0 seconds)
```


Employee

- name: string
- year_of_birth: int
- time: int
- salary: int

- Employee()
- Employee(string, int, int, int)
- recruit()
- timeTrack()
- payroll()
- print()

Contractor

- **Employee():**
 - name = ""
 - year_of_birth = 0
 - time = 0
 - salary = 0
- **Employee(name, year_of_birth, time, salary)**
- **Main:**
 - Declare objEmployee1 call Employee()
 - Declare objEmployee2 call Employee("An", 2000, 100, 50)
 - Print objEmployee1 and objEmployee2

Coordinate

- x: int
- y: int

- `Coordinate()`
- `Coordinate(int,int)`
- `setXY(int, int)`
- `entry()`
- `setX(int)`
- `setY(int)`
- `int getX()`
- `int getY()`
- `move(int,int)`

Constructor

- `Coordinate()`: set $x=0$, $y=0$
- `Coordinate(int, int)`: set x , y = parameters
- `entry()` allow user entry x , y value
- `move(int dx, int dy)`: set new value: $x=x+dx$, $y=y+dx$
- Main
 - Declare $p1$ call `Coordinate()`
 - Declare $p2$ call `Coordinate(5,5)`
 - Print $p1$, $p2$

2. Encapsulation

- In OOP languages, the concept of hiding implementation details of an object is achieved by applying the concept of encapsulation.

Encapsulation

- It is a mechanism which binds code and data together in a class.
- Its main purpose is to achieve data hiding within a class which means:
 - Implementation details of what a class contains need not be visible to other classes and objects that use it.
 - Instead, only specific information can be made visible to the other components of the application and the rest can be hidden.
- By hiding the implementation details about what is required to implement the specific operation in the class, the usage of operation becomes simple.

Access Modifiers

- In Java, the data hiding is achieved by using **access modifiers**.
- **Access Modifiers:**
 - Determine how members of a class, such as instance variable and methods are accessible from outside the class.
 - Decide the scope or visibility of the members.
 - Are of four types:

public

- Members declared as public can be accessed from anywhere in the class as well as from other classes.

private

- Members are accessible only from within the class in which they are declared.

protected

- Members to be accessible from within the class as well as from within the derived classes.

package (default)

- Allows only the public members of a class to be accessible to all the classes present within the same package.
- This is the default access level for all the members of the class.

Access Modifiers

Modifier	Class	Package	Subclass	Global
Public	Yes	Yes	Yes	Yes
Protected	Yes	Yes	Yes	No
Default	Yes	Yes	No	No
Private	Yes	No	No	No

Summary

The class is a logical construct that defines the shape and nature of an object.

Objects are the actual instances of the class and are created using the new operator. The new operator instructs JVM to allocate the memory for the object.

The members of a class are fields and methods. Fields define the state and are referred to as instance variables, whereas methods are used to implement the behavior of the objects and are referred to as instance methods.

Each instance created from the class will have its own copy of the instance variables, whereas methods are common for all instances of the class.

Constructors are methods that are used to initialize the fields or perform startup operations only once when the object of the class is instantiated.

The heap area of memory deals with the dynamic memory allocations for objects, whereas the stack area holds the object references.

Data encapsulation hides the instance variables that represents the state of an object through access modifiers. The only interaction or modification on objects is performed through the methods.

Student Activities

1. Put code define Employee class in others class file but in the same package of main method
Set it's elements in **different accessible** and try to access from itself, instance object (**same package**)
2. Put code define Coordinate class in others package
Set it's elements in **different accessible** and try to access from itself, instance object (**other package**)

Student Activities

Coordinate

- x: int
- y: int

- **Coordinate()**
- **Coordinate(int,int)**
- setXY(int, int)
- entry(int, int)
- setX(int)
- setY(int)
- getX(int)
- getY(int)
- move(int,int)

- Put code define Coordinate class in others package

ĐẠI HỌC FPT CẦN THƠ

