


A large, light blue, stylized watermark of the Java logo is centered in the background. It consists of three concentric, broken circles with a small solid circle in the center, all rendered in a light blue color with a slight gradient.

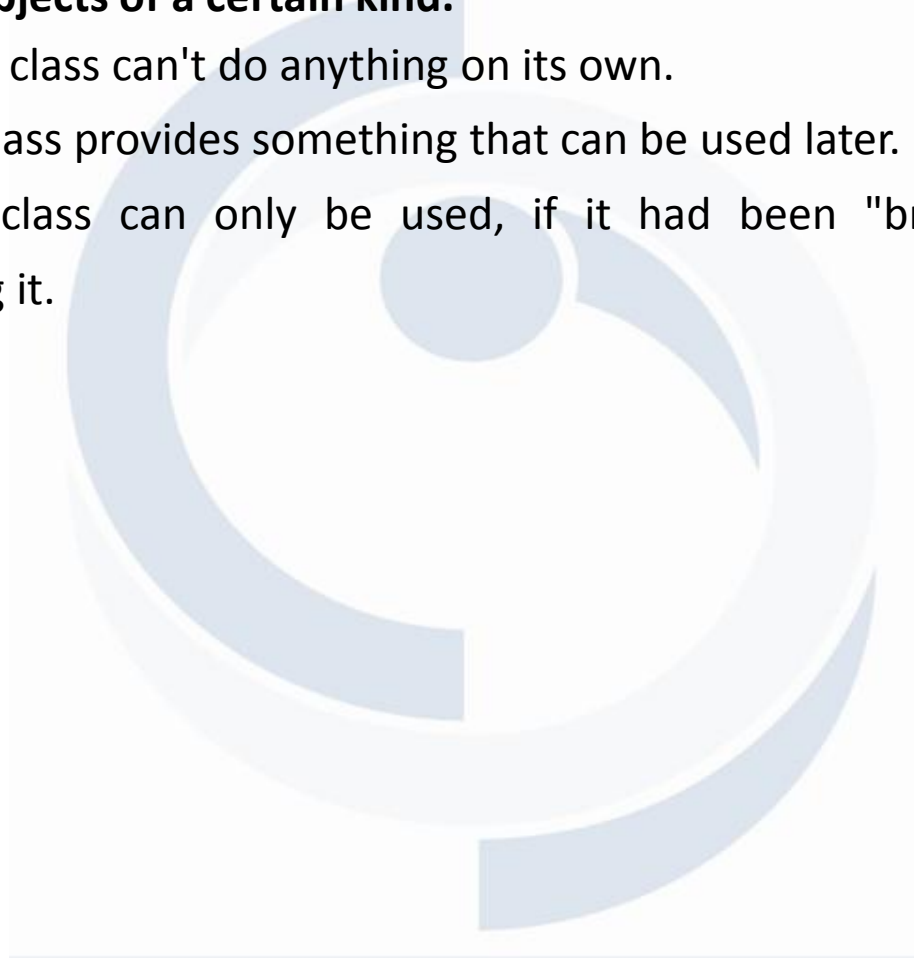
Core Java Training

Topics

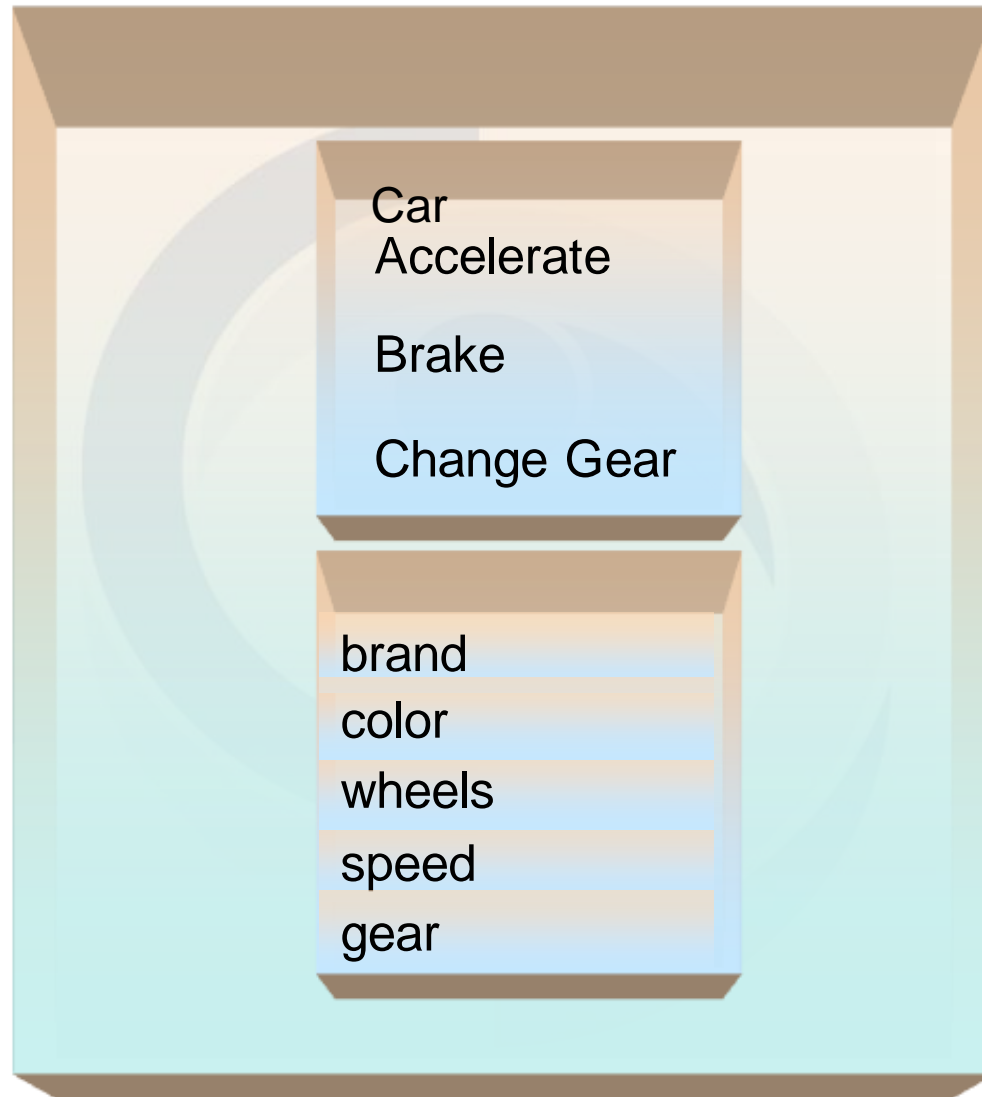
- **Define Class, Methods & Properties**
 - **Understand usage of Packages**
 - **Creating user defined packages**
 - **Managing classes under packages**
 - **Using package members**
 - **Access modifiers**
 - **Working with constructors, Methods**
 - **Creating objects**
- 

What is a class?

- **A class is a blueprint or prototype that defines the variables and the methods common to all objects of a certain kind.**
 - **blueprint:** A class can't do anything on its own.
 - **defines:** A class provides something that can be used later.
 - **objects:** A class can only be used, if it had been "brought to life" by instantiating it.

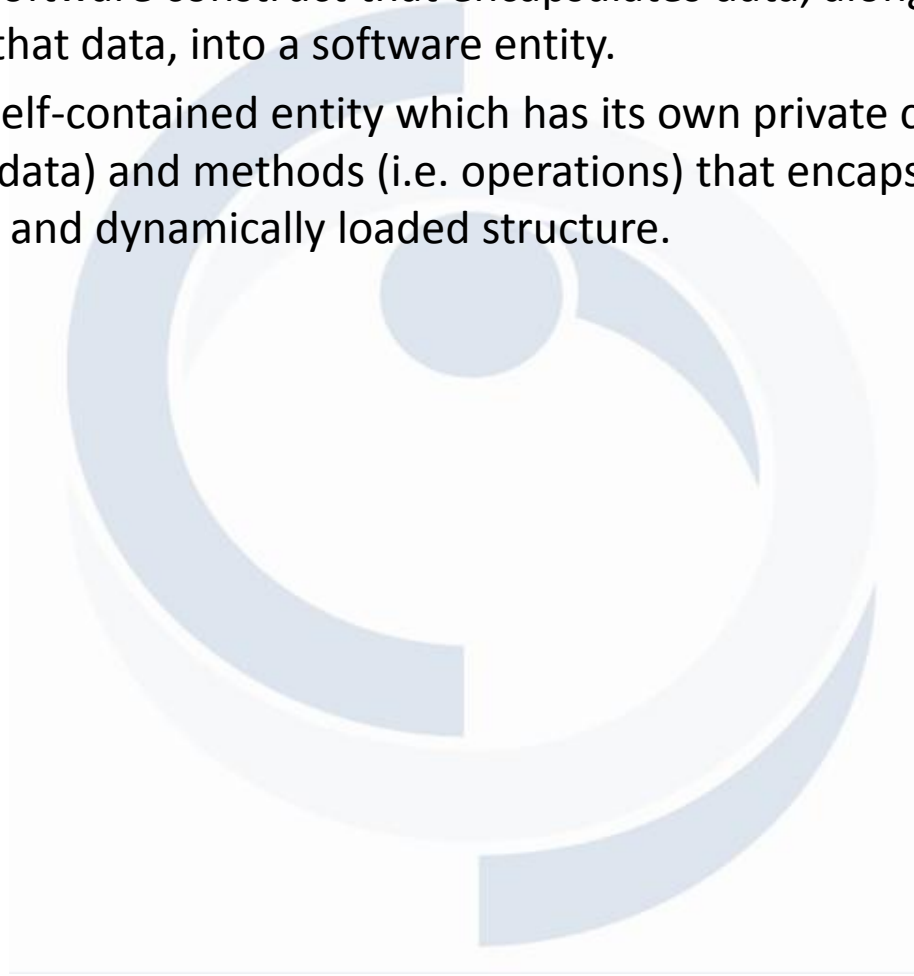


Example: Class



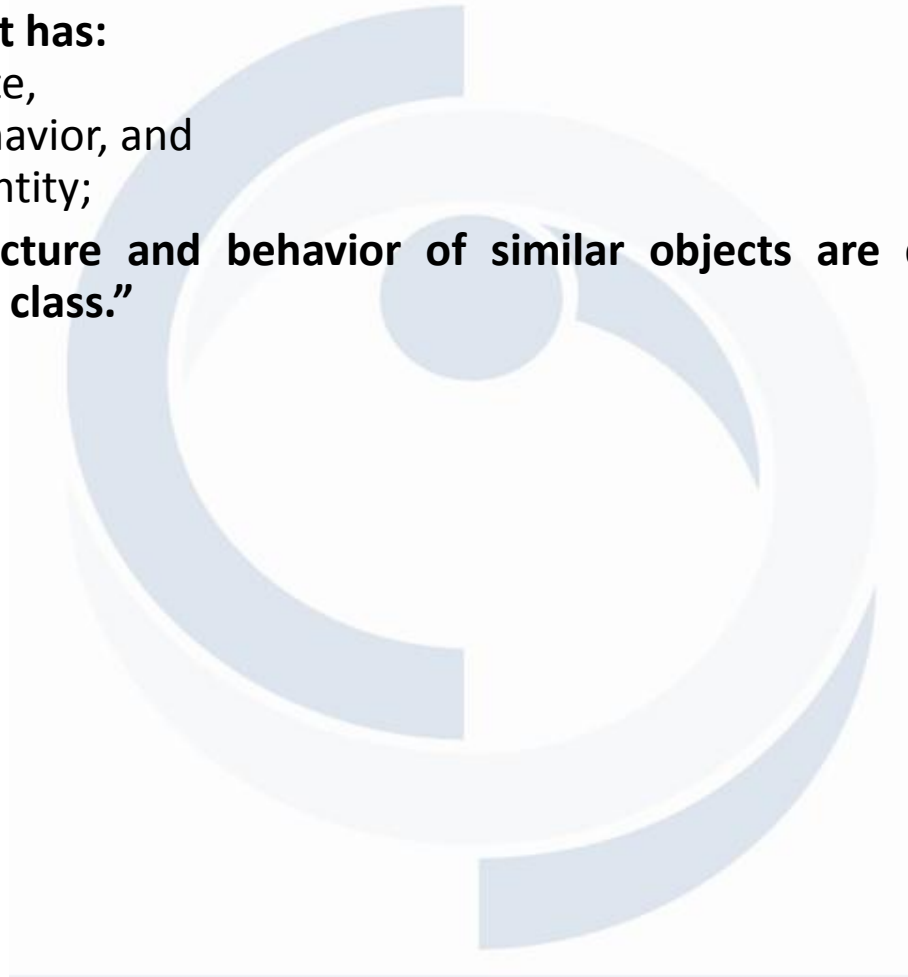
Object

- An object is a software construct that encapsulates data, along with the ability to use or modify that data, into a software entity.
- An object is a self-contained entity which has its own private collection of properties (ie. data) and methods (i.e. operations) that encapsulate functionality into a reusable and dynamically loaded structure.



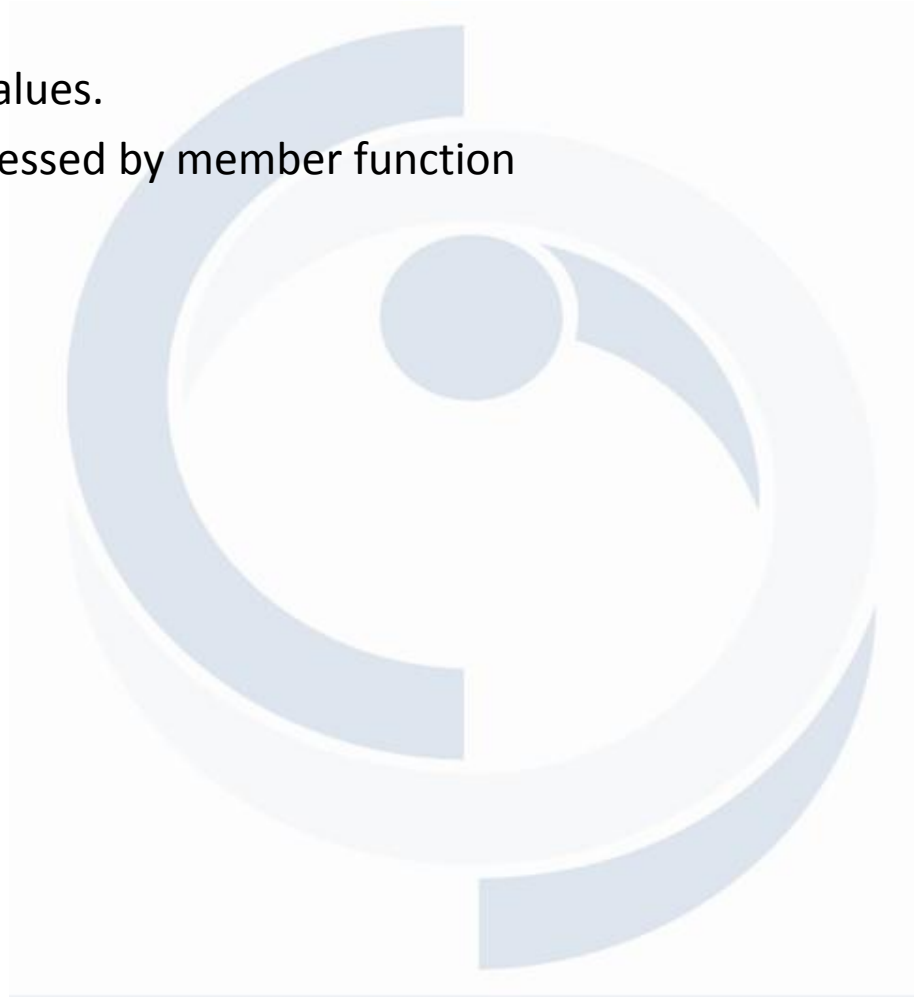
Object (continued)

- **Booch defines an object as: “Something you can do things to”.**
- **An object has:**
 - state,
 - behavior, and
 - identity;
- **The structure and behavior of similar objects are defined in their common class.”**



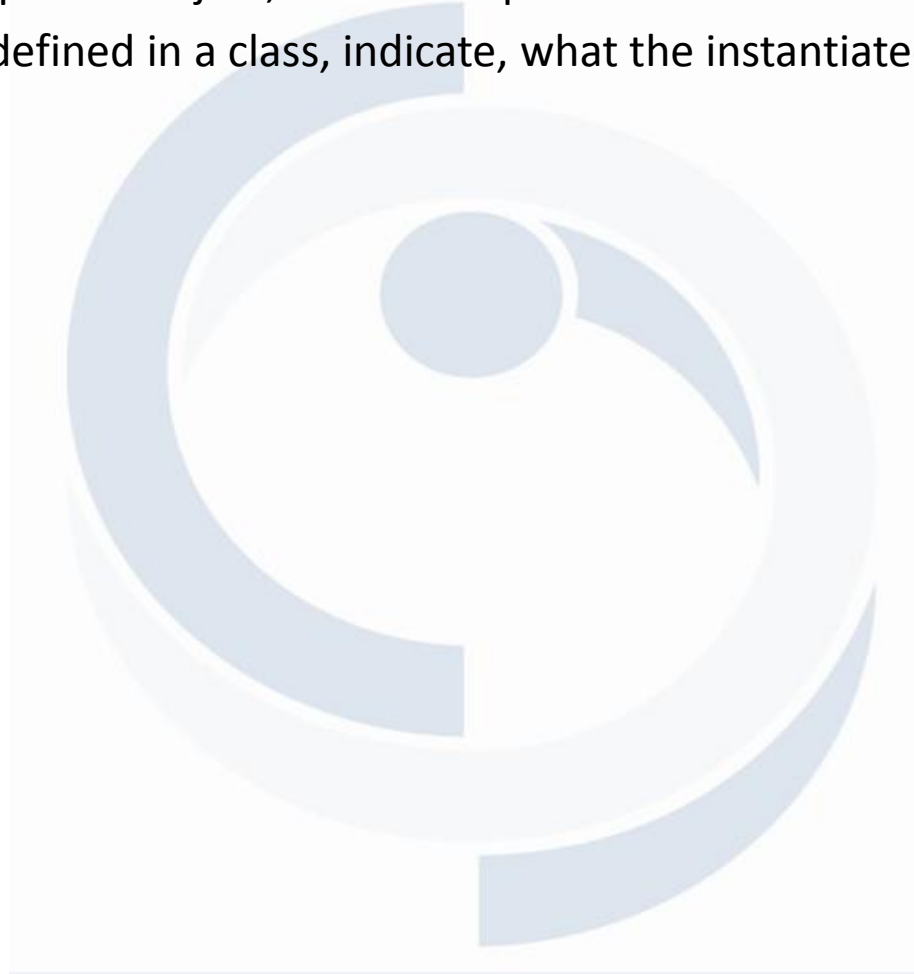
Data Members

- Which store values.
- Which are accessed by member function



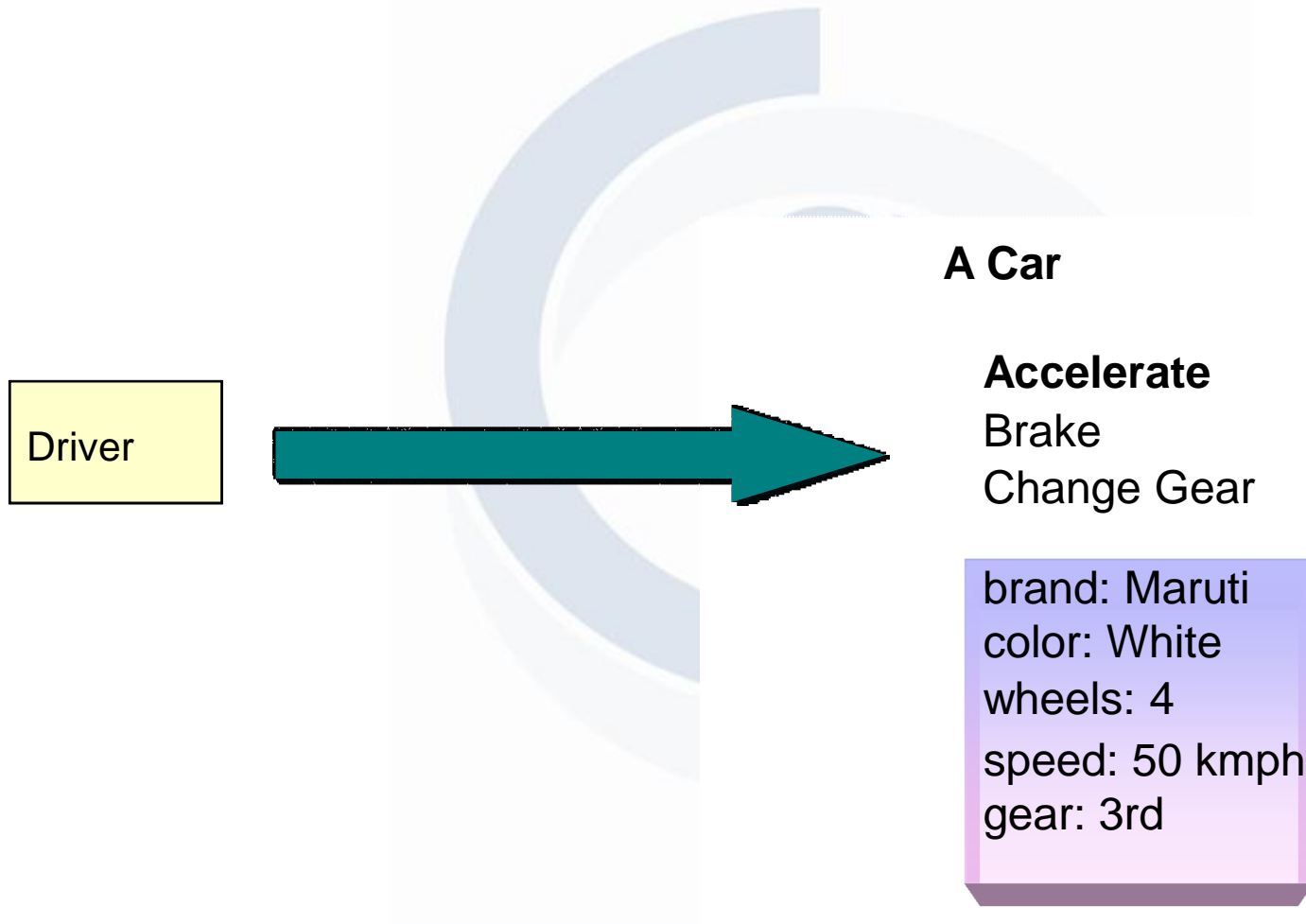
Methods

- An operation upon an object, defined as part of the declaration of a class.
- The methods, defined in a class, indicate, what the instantiated objects are able to do.



Example: Methods

- Driver wants to increase the speed of the car?



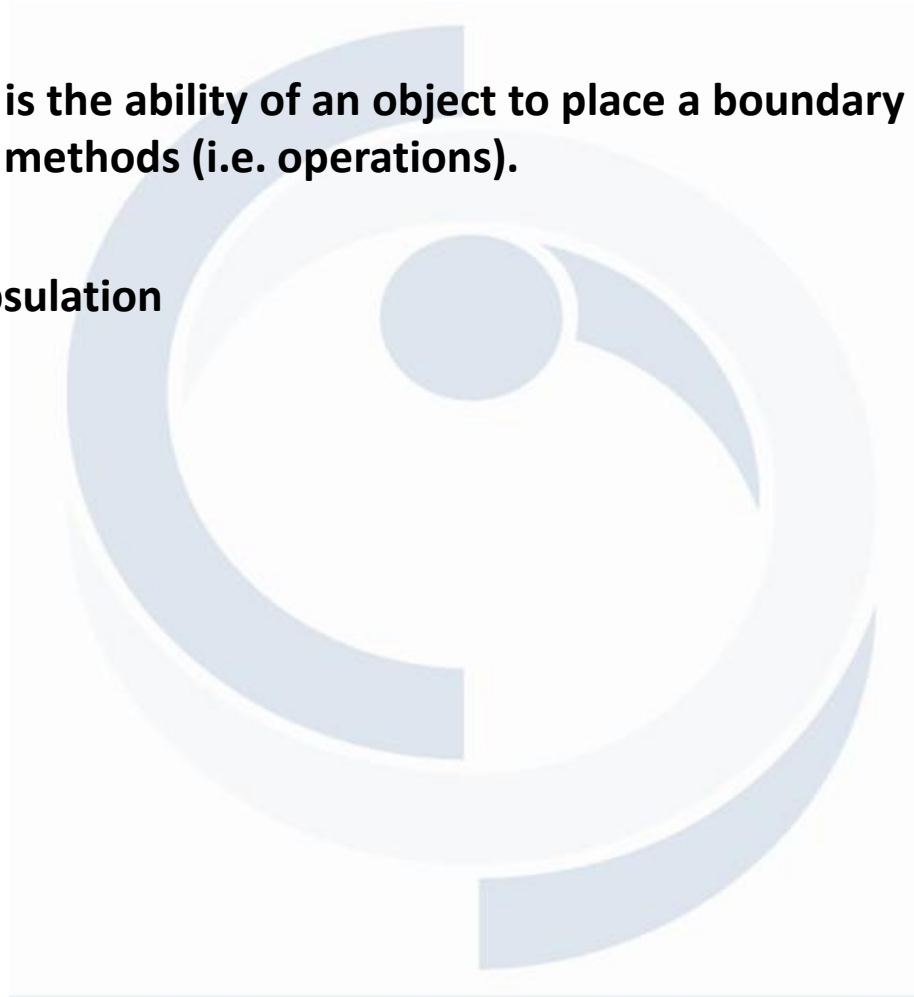
Features of OOPS

- **Encapsulation**
- **Abstraction**
- **Inheritance**
- **Polymorphism**

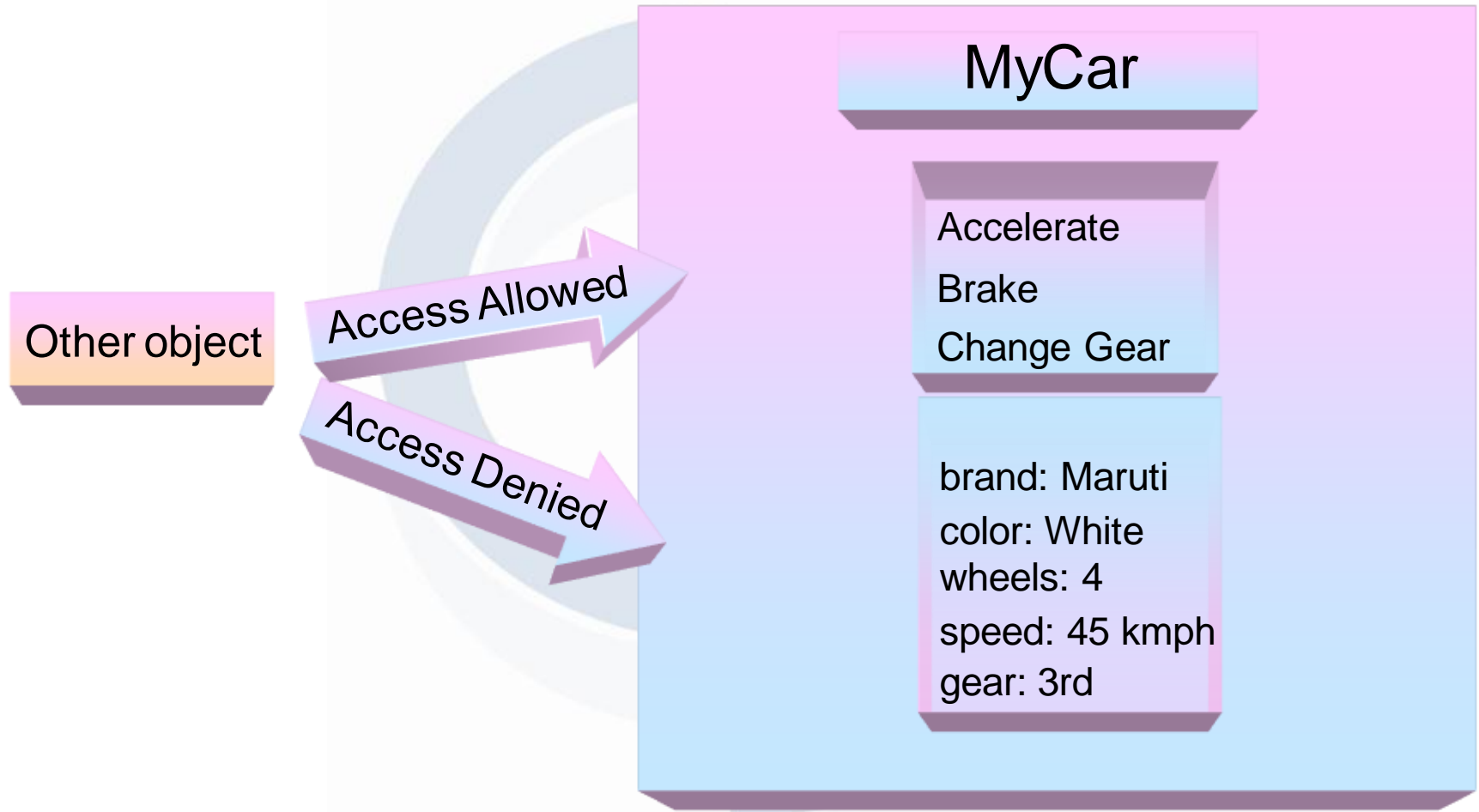


Encapsulation

- **Encapsulation is the ability of an object to place a boundary around its properties (i.e. data) and methods (i.e. operations).**
- **Object - encapsulation**



Example: Encapsulation



Structure of Java application (continued)

Class Example

```
class Car
{
    String brand;
    String color;
    int wheels;
    int speed;
    int gear;
    void accelerate(){}    Method Name
    void brake(){}
    void changeGear(){}
}
```

Steps to create Object

Creation Of Object

- `Car myCar1; // declaration of reference variable no memory allocated`
- `myCar = new Car(); // object is created Memory is allocated`
- `Car myCar1= new Car();//`

Structure of Java application (continued)

Object of Classes (Continued)

Object type, matches the class name.

Instantiating a new object

```
Car MyCar1 = new Car();
```

Class name or the default constructor

An instance named MyCar1

Tells compiler to call constructor to get new object (at a new memory location)

Structure of Java application (continued)

Accessing Data Members of a Class

- Dot Operator(.) is used to access the data members of a class outside the class by specifying the data member name or the method name.

```
MyCar2.brand = "Tata Indica";  
MyCar2.color = "Dark Red";
```

```
MyCar1.brand = "Hyundai Santro";  
MyCar1.color = "Silver";
```

```
MyCar3.brand = "Maruti Esteem";  
MyCar3.color = "Red";
```


Constructors

- A constructor is a special method that initializes the instance variables. The method is called automatically when an object of that class is created.
- A constructor method has the same name as the that of the class.
- A constructor is most often defined with the accessibility modifier “public” so that every program can create an instance but is not mandatory.
- A constructor is provided to initialize the instance variables in the class when it is called.

Constructor (continued)

- If no constructor is provided, a default constructor is used to create instances of the class. A default Constructor initializes the instance variables with default values of the data types.
- If at least one constructor is provided, default constructor is not provided by JVM.
- A class can have multiple constructors.

```
class Car {  
    String brand;  
    String color;  
    int wheels;  
    int speed;  
    int gear;  
    public Car()  
    {brand="NoBrand";  
    color="NoColor";  
    wheels=4;  
    speed=80;  
    }
```

```
    public Car(String b, String c, int w,  
                int s, int g)  
    {  
        brand=b;  
        color=c;  
        wheels=w;  
        speed=s;  
    }  
    void accelerate(){}  
    void brake(){}  
    void changeGear(){}  
}
```

The *this* reference

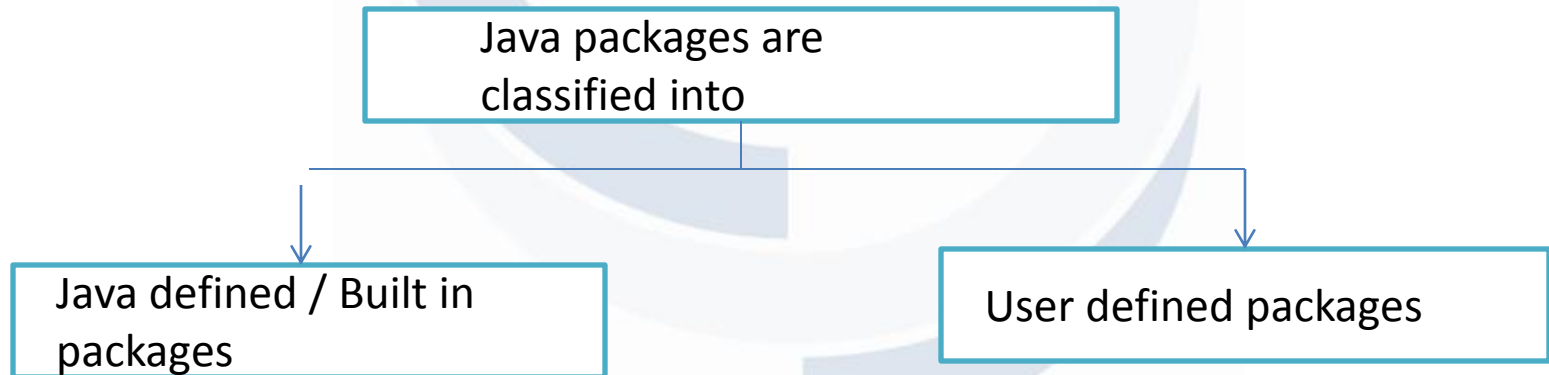
- **“this” is a reference to the current object.**
- **this is used to**
 - refer to the current object when it has to be passed as a parameter to a method.
 - `otherobj1.anymethod(this);`
 - refer to the current object when it has to be returned in a method.
 - Refer the instance variables if parameter names are same as instance variables to avoid ambiguity.
 - `public void method1(String name){`
 - `this.name=name;`
 - `}`

The *this* reference (continued)

- The `this` keyword included with parenthesis i.e. `this()` with or without parameters is used to call another constructor. The default construct for the `Car` class can be redefined as below
 - `public Car(){`
 - `this("NoBrand",""NoColor",4,0);`
 - `}`
- The `this()` can be called only from a constructor and must be the first statement in the constructor

Understanding usage of packages

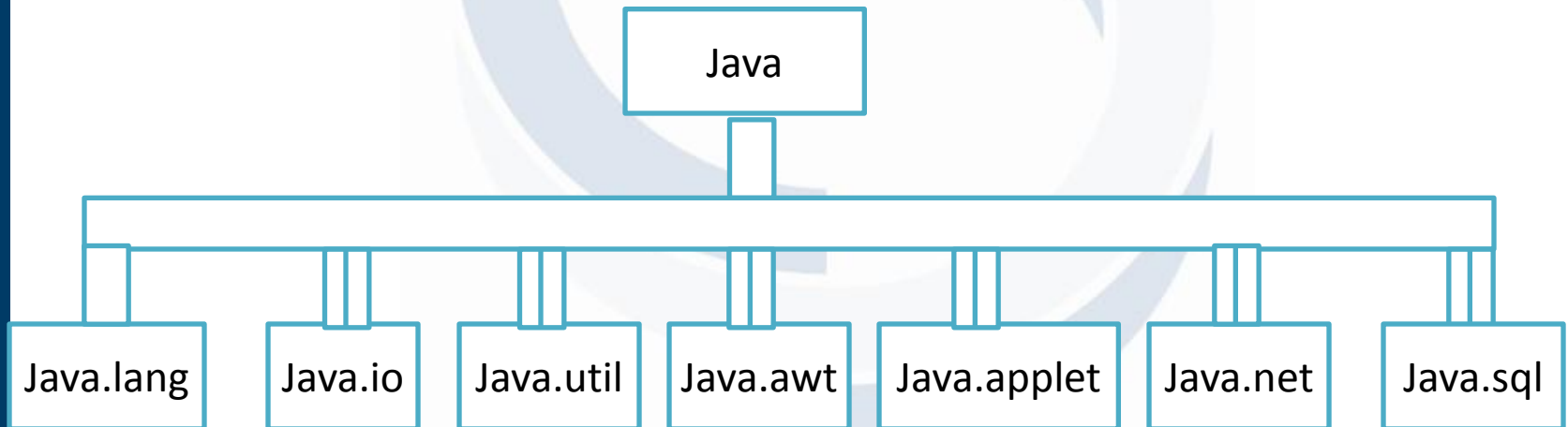
- "A package is a namespace that organizes the a set of related classes and interfaces. It also defines the scope of a class. An application consists of thousands of classes and interfaces, and therefore, it is very important to organize them logically into packages.
- By definition, package is a grouping of related types providing access protection and name space management."
- Packages enable you to organize the class files provided by Java.



Packages in Java

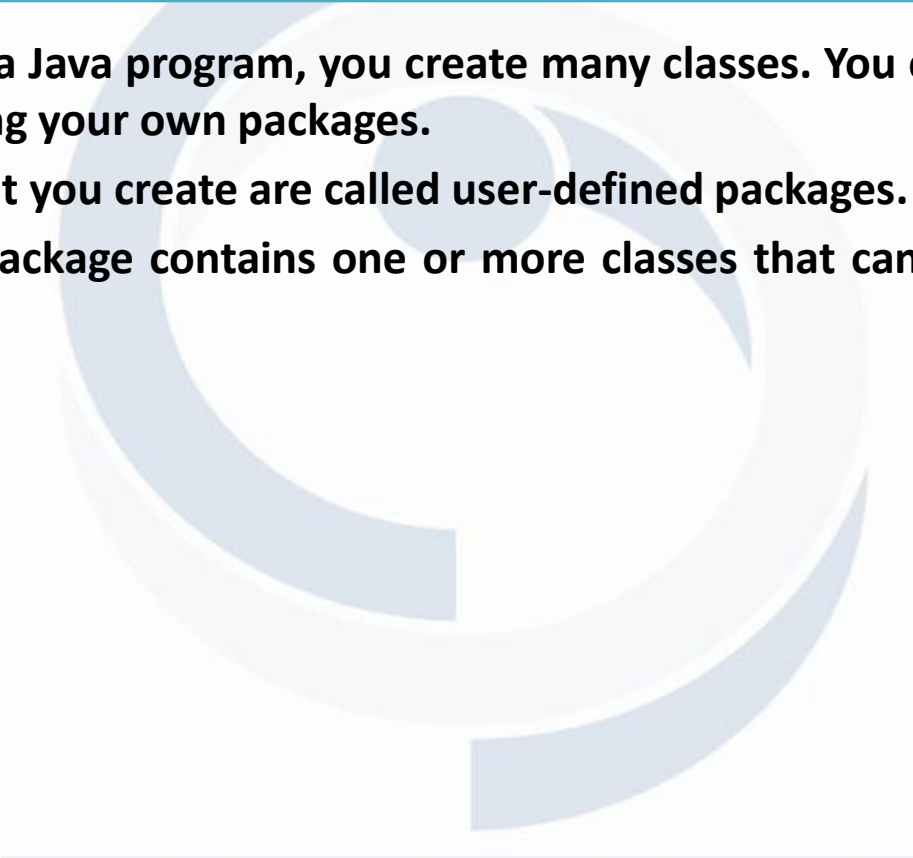
Hierarchy of Java Packages

The packages in Java can be nested. The 'Java' is the root package for all the packages of Java API. Below is the partial hierarchy of Java API.



The classes of a package are qualified with package name

User Defined Packages

- **When you write a Java program, you create many classes. You can organize these classes by creating your own packages.**
 - **The packages that you create are called user-defined packages.**
 - **A user-defined package contains one or more classes that can be imported in a Java program.**
- 

User defined packages (continued)

Syntax & Example

- Creating a user-defined package

```
package <package_name>
// Class definition
public class <classname1>
{
    // Body of the class.
}
```

```
package land.vehicle;
public class Car
{
    String brand;
    String color;
    int wheels;
}
```


User defined packages (continued)

Importing Packages

- You can include a user-defined package or any Java API using the import statement.
- The following syntax shows how to implement the user-defined package, vehicle from land in a class known as MarutiCar
 - `import land.vehicle.Car;`
 - `public class MarutiCar extends Car`
 - `{`
 - `// Body of the class.`
 - `}`

Packages in Java (continued)

Built in Java Packages

Java Package Name	Description
java.lang	Includes various classes, such as Object, System, Thread etc.
java.io	Includes all Input-Output Stream related classes.
java.util	Provides various classes that support Date, Collection.
java.sql	Provides API for Database operation

Packages in Java (continued)

Built in Java Packages

Java Package Name	Description
java.awt	Providing Supporting classes for Graphic User Interface components.
java.applet	Provides the Applet class to create web based application.
java.net	Includes classes that support network programming such as Socket, DatagramSocket.

Access modifiers

Access Modifiers

An attribute that determines whether or not a class member is accessible in an expression or declaration.

Type of Access Modifier

public

Members are accessible to the class and to other classes.

private

Members are accessible only to the class.

protected

Members are accessible only to the class and its subclass(es).

package /
friendly

Members are accessible only to the class and other classes within that package. This is the default access specifier.

	In Same Package			In Different Package	
	Class	subclass	non subclass	subclass	non subclass
private	Y	N	N	N	N
Default	Y	Y	Y	N	N
Protected	Y	Y	Y	Y	N
Public	Y	Y	Y	Y	Y

Modifiers

Modifiers

- Determines how data members and methods are used in other classes.

Types of Modifiers

- | |
|-----------------|
| 1. final |
| 2. static |
| 3. abstract |
| 4. synchronized |

Types of modifiers

1. final

1.a) A final data members cannot be modified.

1.b) A final method cannot be overridden in the subclass.

1.c) A class declared as final cannot be inherited .

Types of modifiers (continued)

2. static

2.a) Used to define data members and methods that belongs to a class and not to any specific instance of a class.

2.b) Methods and data members can be called without instancing the class.

2.c) A static member exists when no objects of that class exist.

Types of modifiers (continued)

3. abstract

3.a) Used to declare class that provides common behavior across a set of subclasses.

3.b) Abstract class provides a common root for a group of classes.

3.c) An abstract keyword with a method does not have any definition.

Types of modifiers (continued)

4. synchronized

Controls the access to a block in a multithreaded environment.

Packages in Java (continued)

Built in Java Packages

Java Package Name	Description
java.lang	Includes various classes, such as Object, System, Thread etc.
java.io	Includes all Input-Output Stream related classes.
java.util	Provides various classes that support Date, Collection.
java.sql	Provides API for Database operation

Packages in Java (continued)

Built in Java Packages

Java Package Name	Description
java.awt	Providing Supporting classes for Graphic User Interface components.
java.applet	Provides the Applet class to create web based application.
java.net	Includes classes that support network programming such as Socket, DatagramSocket.

Method overloading

Java allows to define several methods with same name within a class

- **Methods are identified with the parameters set based on:**
 - the number of parameters
 - types of parameters
 - order of parameters.
- **Compiler selects the proper method.**
- **This is called as Method overloading.**
- **Method overloading is used perform same task with different available data**
 - **Example:**
 - `int sum(int a,int b) { return a+b;}`
 - `int sum(int a,int b,int c) { return a+b+c;}`
 - `float sum(float a, float b, float c) { return a+b+c;}`