Mphasis
The Next Applied

Java Foundation – S1

Object Oriented Programming

Presentation By : V Sindhu

SME - Java

Dos and Don'ts

- Login to GTW session on Time
- Login with your Mphasis email ID
- Use the question window for asking any queries

Day - 2

Object Oriented Programming

Agenda

- Define Class, Methods & Properties
- Working with constructors
- Use of “this” keyword
- Method overloading
- Understand usage of Packages
- Creating user defined packages
- Managing classes under packages
- Using package members
- Access modifiers
- Significance of import keyword
- Different modifiers (final, static, abstract etc.)

Objectives

- After completion of this module, you should be able to:
- Understand the need for object-oriented programming
  - ✓ About Procedural Programming
  - ✓ Working of Procedural Programming
- Compare OOPS with procedural programming
- Identify the advantages of object-oriented programming
- Identify classes and objects
- Explain the features of OOP

About Procedural Programming

- Procedural Programming can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure.
- Procedures, also known as routines, subroutines or functions, simply consist of a series of computational steps to be carried out.
- During a program's execution, any given procedure might be called at any point, including by other procedures or itself.
- Languages used in Procedural Programming:
- FORTRAN, ALGOL, COBOL, BASIC, Pascal and C.

Working with procedural programming

- In procedural programming, a program consists of data and modules/procedures that operate on the data. The two are treated as separate entities.
- In the object-oriented programming (OOP) paradigm, however, a program is built from objects. An object is an instance of a class, which is an encapsulation of data (called fields) and the procedures (called methods) that manipulate them.
- In most, but not all, cases, the fields can only be accessed or modified through the methods.
- An object therefore is like a miniature program or a self-contained component, which makes the OOP approach more modularized and thus easier to maintain and extend.

Understand the need for object-oriented programming

- Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism etc in programming.
- The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

Some of the important OOPS concepts

- Class
- Object
- Method and method passing
- Pillars of OOPS
  - Abstraction
  - Encapsulation
  - Inheritance
  - Polymorphism
    - Compile-time polymorphism
    - Run-time polymorphism

Compare OOPS with procedural programming

- Procedural Programming can be defined as a programming model which is derived from structured programming, based upon the concept of calling procedure.
- Object-oriented programming can be defined as a programming model which is based upon the concept of objects. Objects contain data in the form of attributes and code in the form of methods. In object-oriented programming, computer programs are designed using the concept of objects that interact with the real world.
- Languages used in Object-Oriented Programming:
- Java, C++, C#, Python, PHP, JavaScript, Ruby, Perl, Objective-C, Dart, Swift, Scala.

Compare OOPS with procedural programming

Procedural Oriented Programming	**Object-Oriented Programming**
In procedural programming, the program is divided into small parts called ***functions***.	In object-oriented programming, the program is divided into small parts called ***objects***.
Procedural programming follows a ***top-down approach***.	Object-oriented programming follows a ***bottom-up approach***.
There is no access specifier in procedural programming.	Object-oriented programming has access specifiers like private, public, protected, etc.
Adding new data and functions is not easy.	Adding new data and function is easy.

Compare OOPS with procedural programming

In procedural programming, there is no concept of data hiding and inheritance.	In object-oriented programming, the concept of data hiding, and inheritance is used.
In procedural programming, the function is more important than the data.	In object-oriented programming, data is more important than function.
Procedural programming is based on the ***unreal world***.	Object-oriented programming is based on the ***real world***.
Procedural programming is used for designing medium-sized programs.	Object-oriented programming is used for designing large and complex programs.
Examples: C, FORTRAN, Pascal, Basic, etc.	**Examples:** C++, Java, Python, C#, etc.

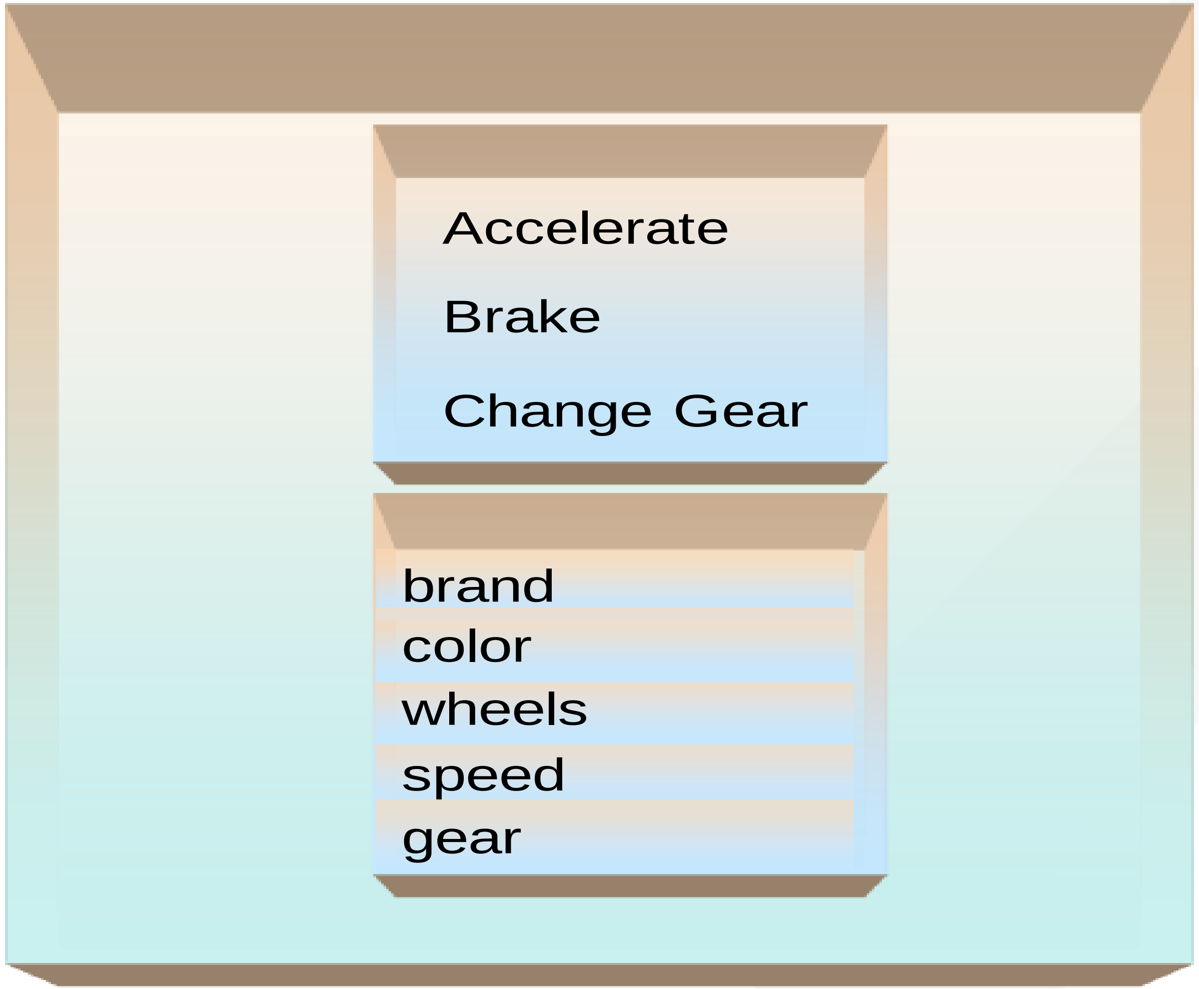
Identify the advantages of object-oriented programming

- Modularity for easier troubleshooting. When working with object-oriented programming languages, you know exactly where to look when something goes wrong.
- Reuse of code through inheritance.
- Flexibility through polymorphism.
- Effective problem solving.

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 MyCar



Accelerate

Brake

Change Gear

brand

color

wheels

speed

gear

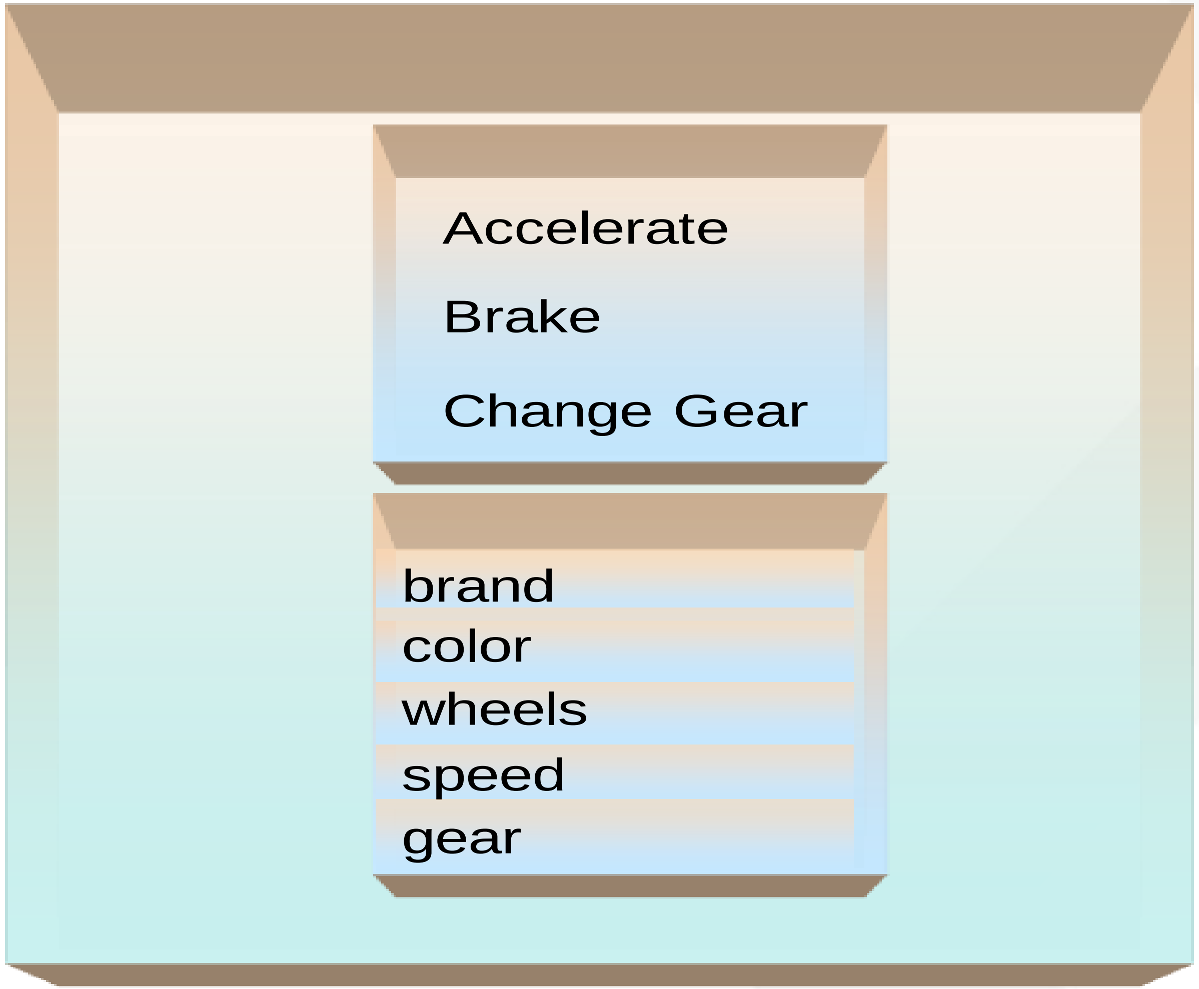
Object

- 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 (i.e., 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
  - identity
- The structure and behavior of similar objects are defined in their common class.

Example: Object



Accelerate

Brake

Change Gear

brand

color

wheels

speed

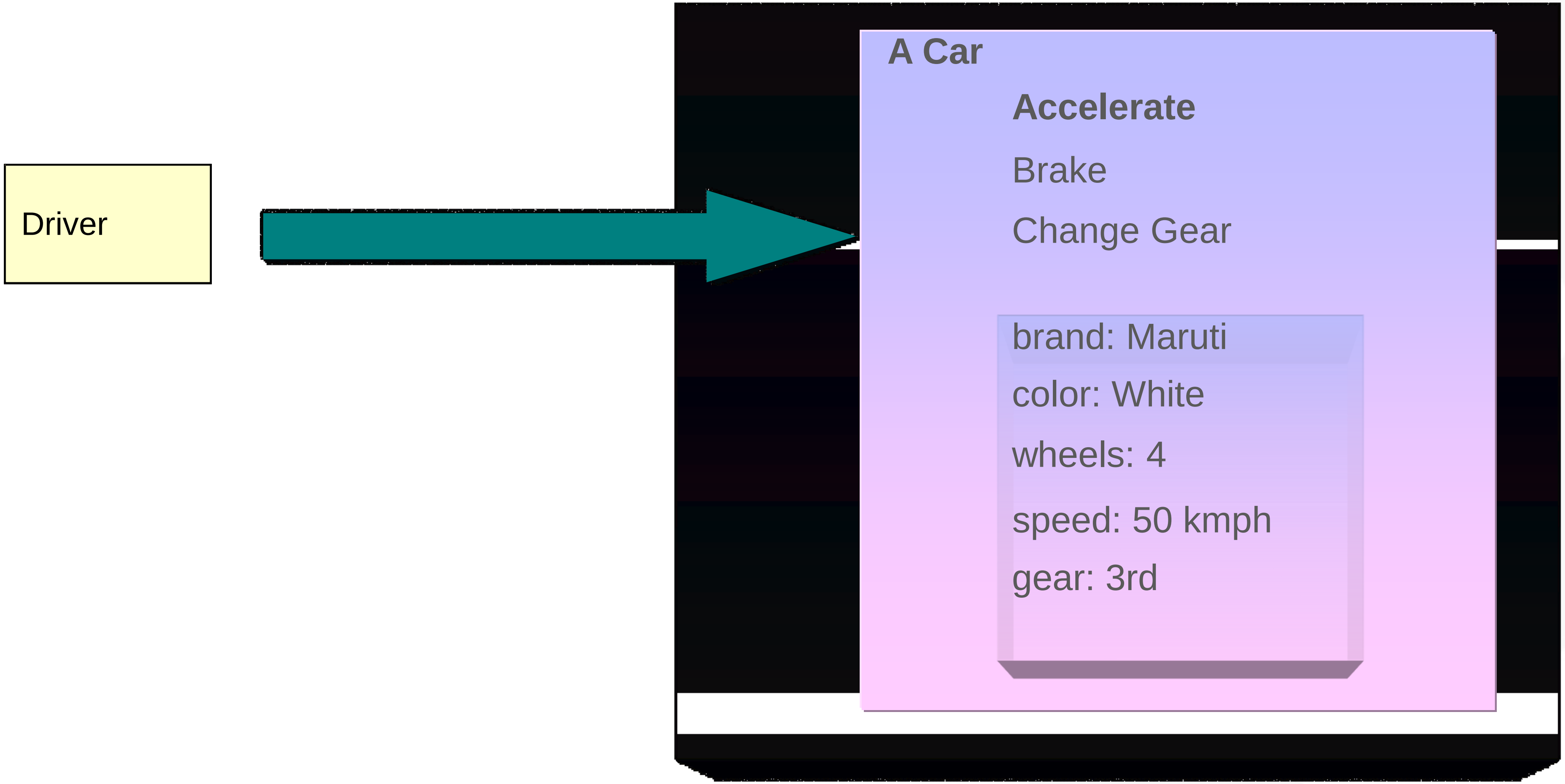
gear

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 can do.

Example: Methods

- Driver wants to increase the speed of the car?



Driver

A Car

Accelerate

Brake

Change Gear

brand: Maruti

color: White

wheels: 4

speed: 50 kmph

gear: 3rd

Object oriented program (OOPS)

- An Object-Oriented Program consists of a group of cooperating objects, exchanging messages, for the purpose of achieving a common objective.
- Benefits of OOPS:
  - Real-world programming
  - Reusability of code
  - Modularity of code
  - Resilience to change
  - Information hiding

Pillars of OOPS

Encapsulation

Abstraction

Inheritance

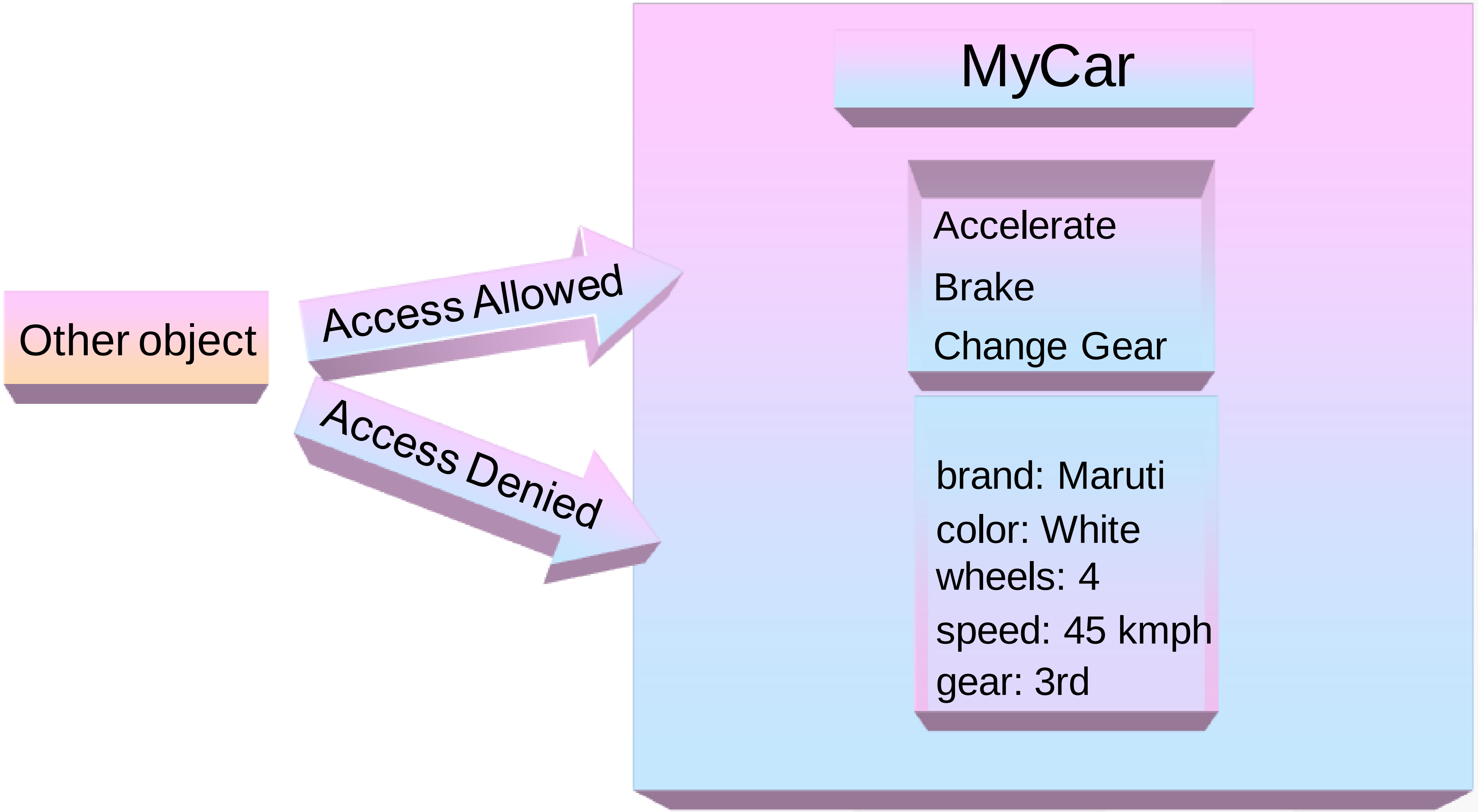
Polymorphism

Encapsulation

- Encapsulation is the ability of an object to place a boundary around its properties (i.e., data) and methods (i.e., operations).
- Grady Booch, defined the encapsulation feature as:

"Encapsulation is the process of hiding all of the details of an object that do not contribute to its essential characteristics."

Example: Encapsulation



Abstraction

- “An Abstraction denotes the essential characteristics of an object that distinguishes it from all other kinds of objects and thus provides crisply defined conceptual boundaries, relative to the perspective of the viewer.”

Encapsulation hides the irrelevant details of an object and **Abstraction** makes only the relevant details of an object visible.

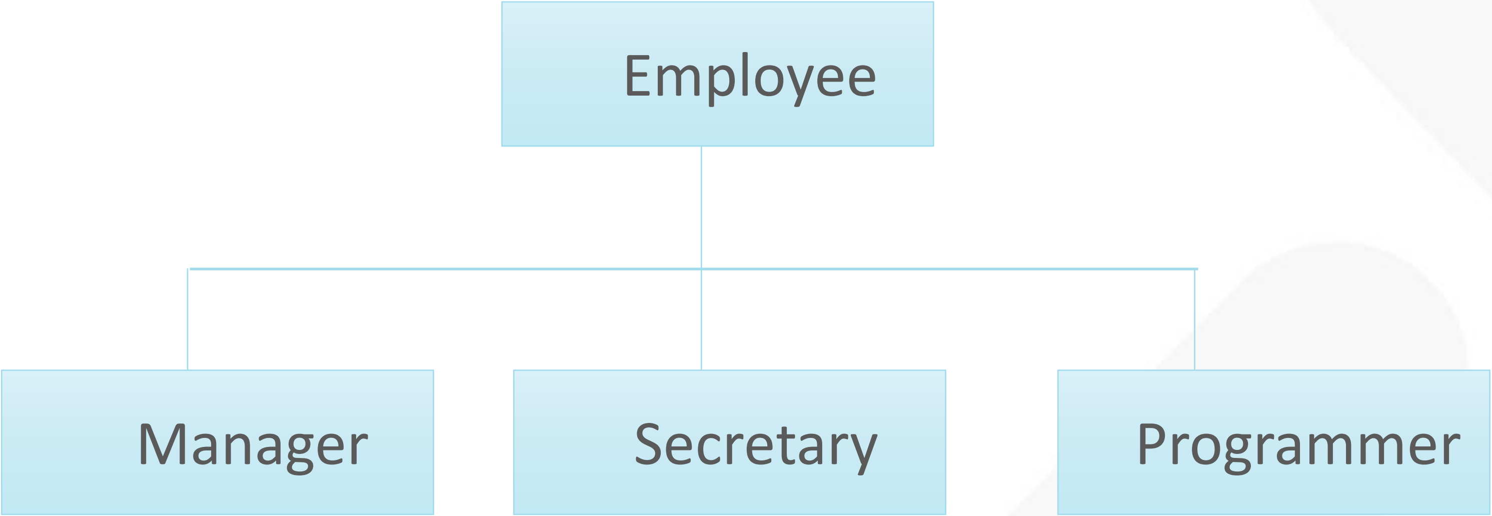
Inheritance

- Inheritance is the capability of a class to use the properties and methods of another class while adding its own functionality.
- Enables you to add new features and functionality to an existing class without modifying the existing class.

Superclass and Subclass

- A superclass or parent class is the one from which another class inherits attributes and behavior.
- A subclass or child class is a class that inherits attributes and behavior from a superclass.

Inheritance (continued)



Employee

Manager

Secretary

Programmer

Types of inheritance

1. Single inheritance : Only Two classes are involved.

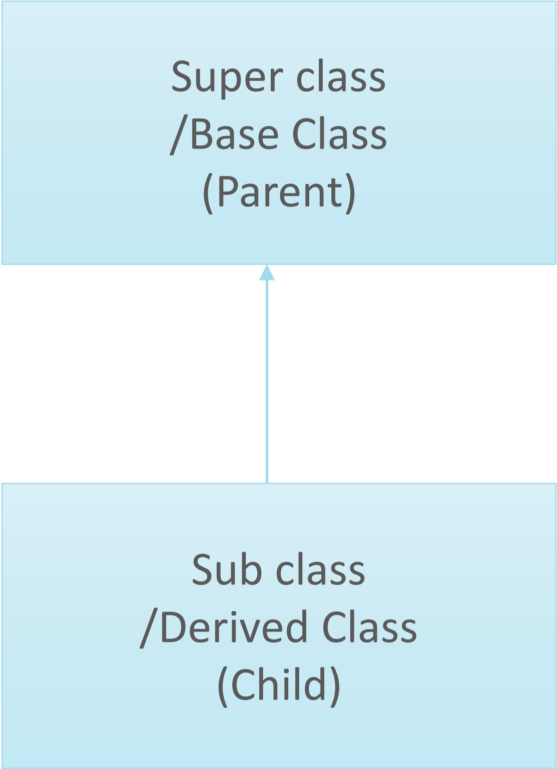
2. Hierarchical Inheritance : N number of classes involved in a Tree like structure

3. Multilevel Inheritance : Multiple levels of Single inheritance

- Multiple inheritance -> Not Supported

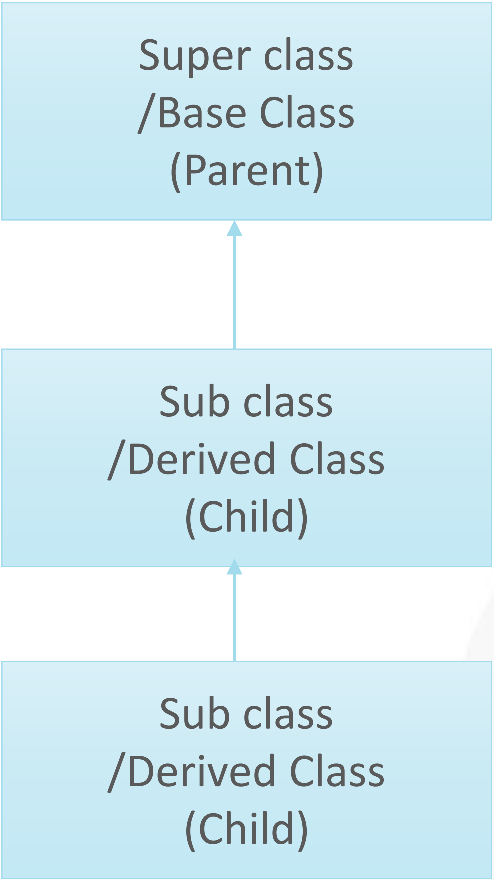
Single inheritance

- Subclass is derived from only one superclass.



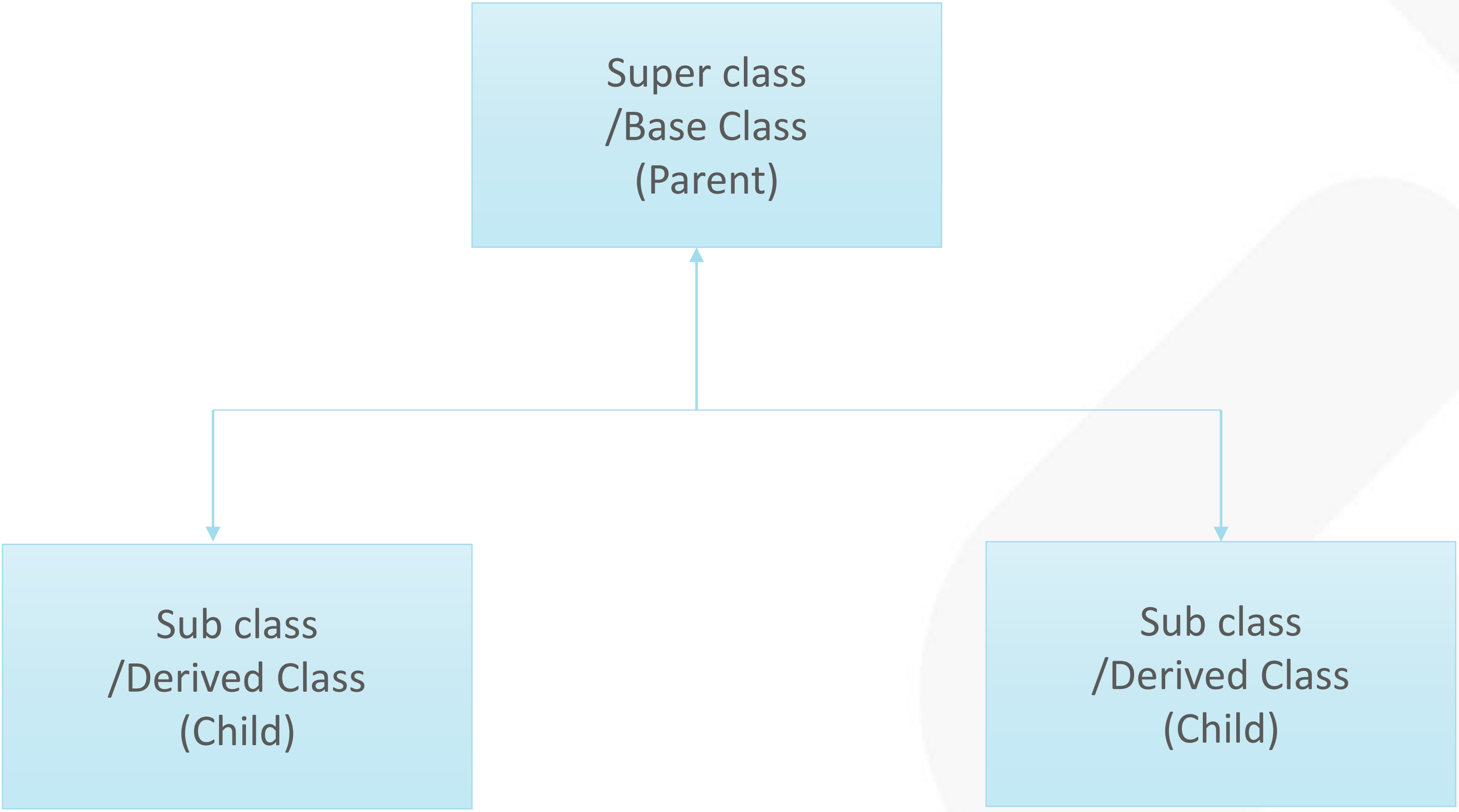
Multilevel Inheritance

- Inherits the properties of another subclass.



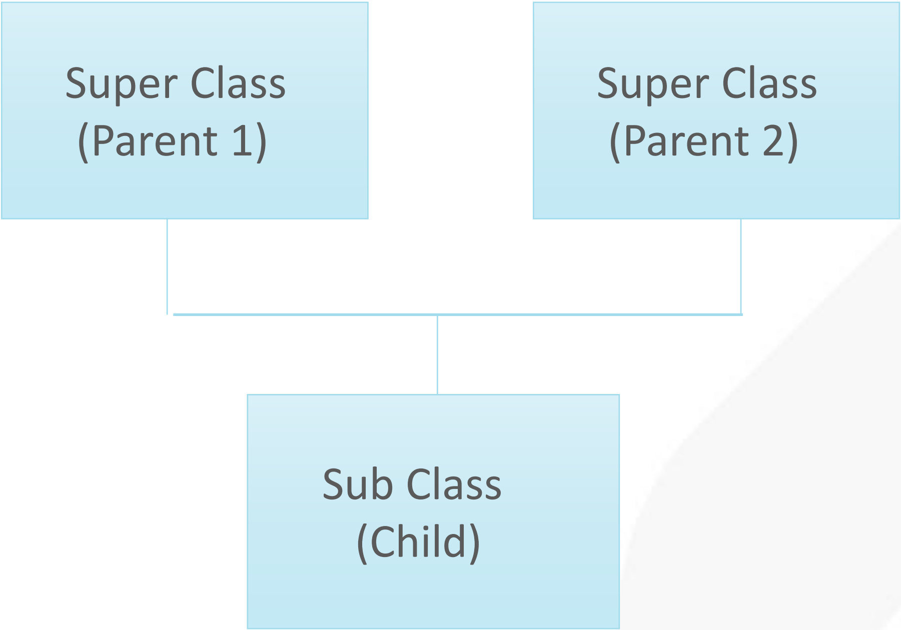
Hierarchical Inheritance

- Number of subclasses inherits from a superclass.



Multiple inheritance

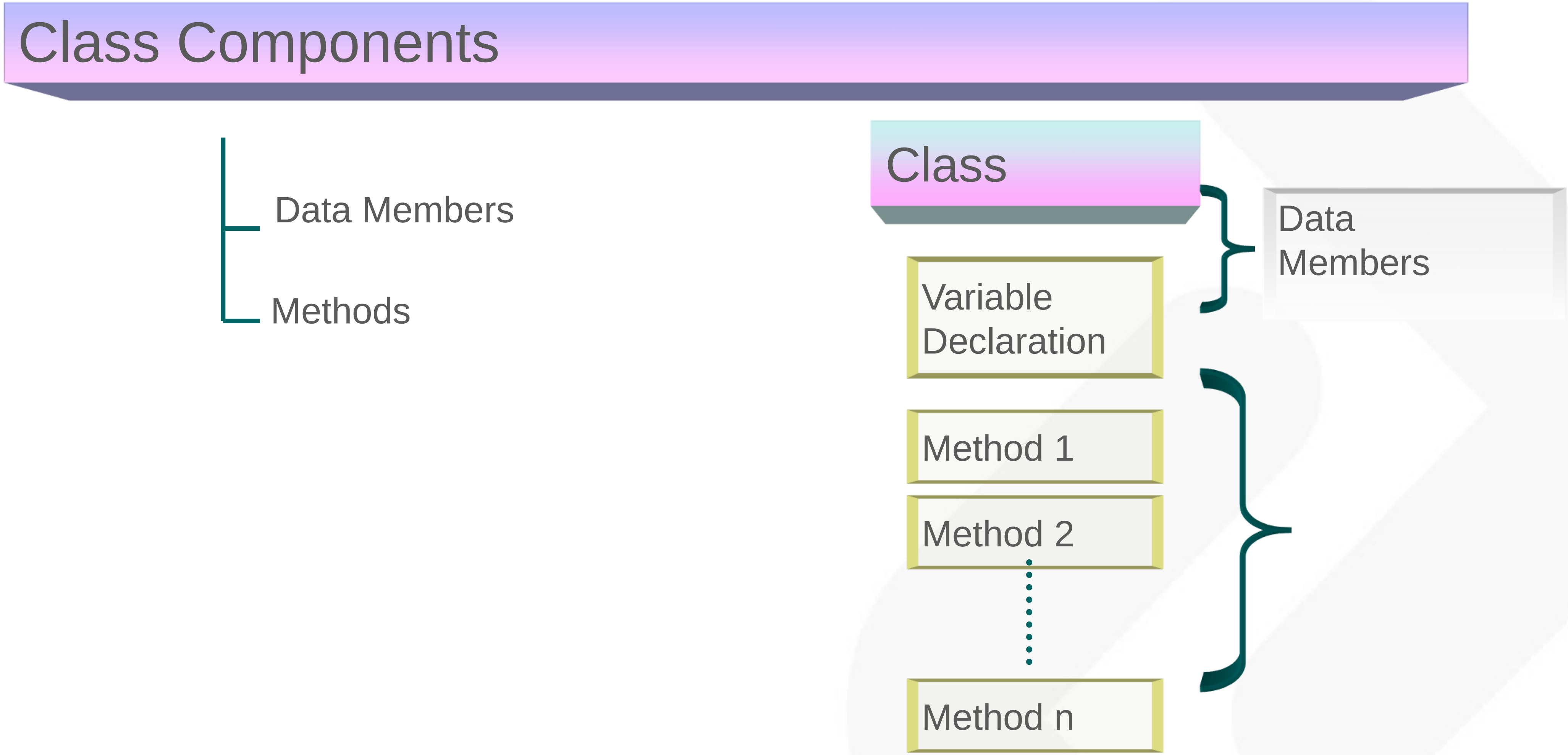
- A subclass is derived from more than one super class.
- Not supported in Java



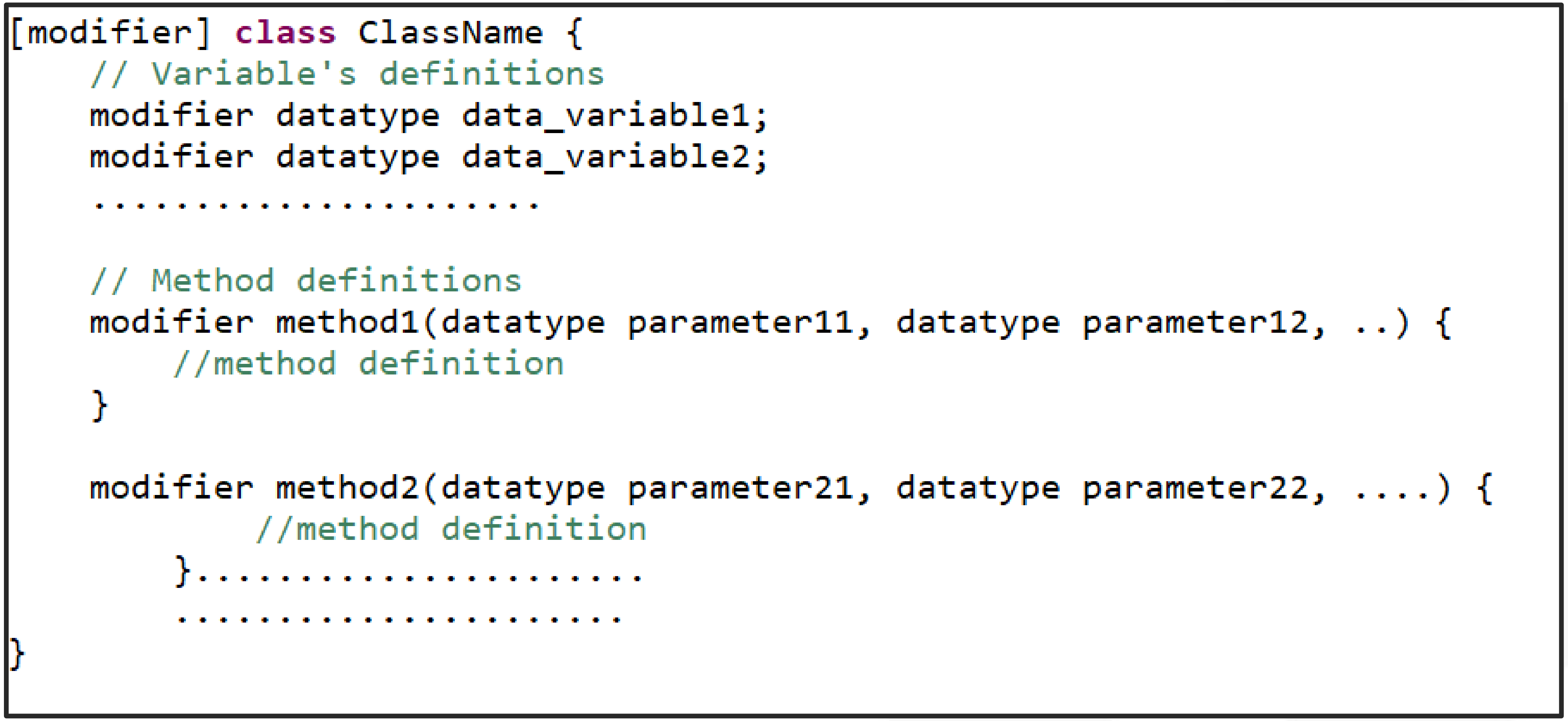
Polymorphism

- Derived from two Latin words - Poly, which means many, and morph, which means forms.
- It is the capability of an action or method to do different things based on the object that it is acting upon.
- In object-oriented programming, polymorphism refers to a programming language's ability to process objects differently depending on their data type or class.

Structure of Java application



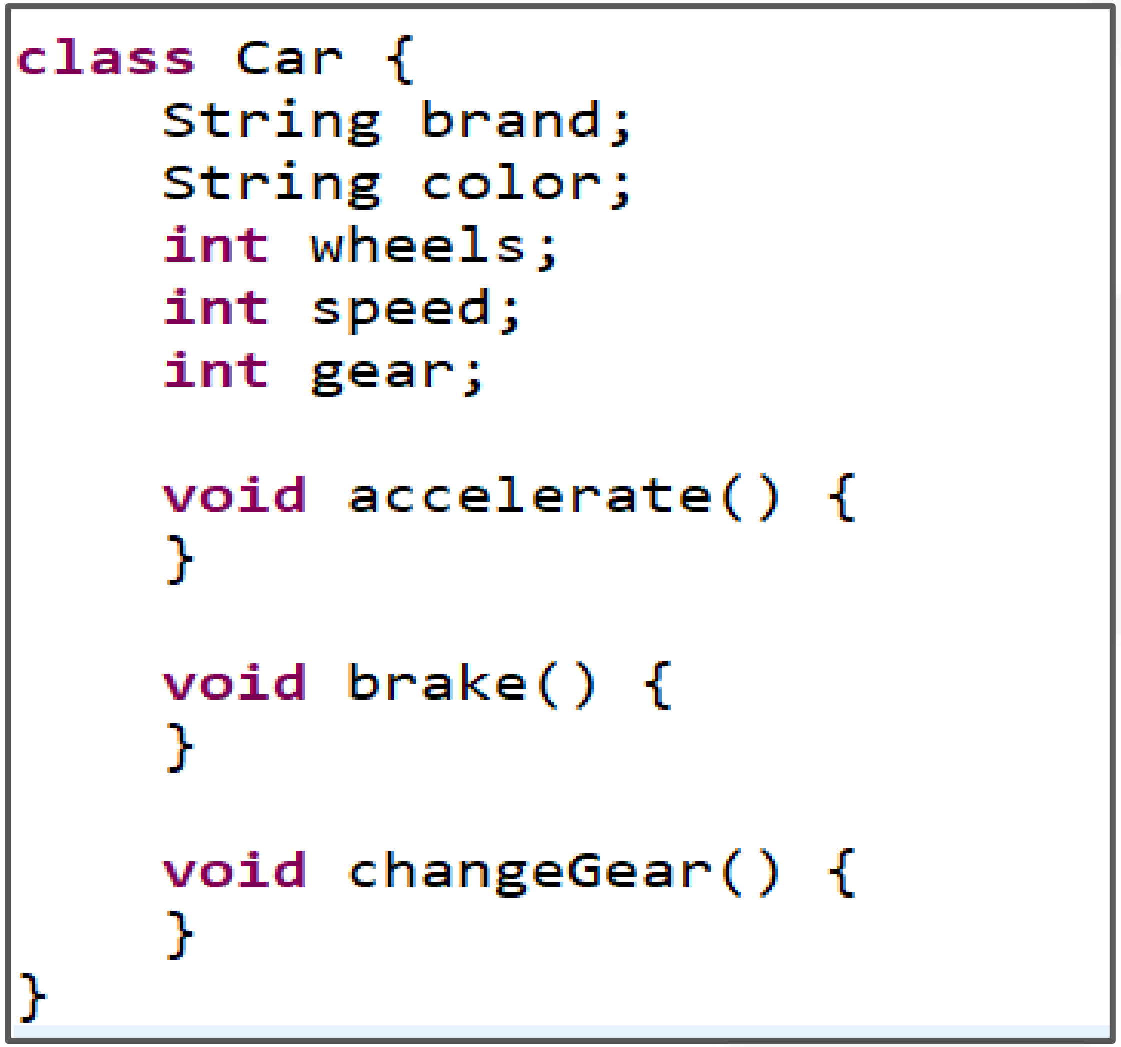
Java class definition: Syntax



```
[modifier] class ClassName {
    // Variable's definitions
    modifier datatype data_variable1;
    modifier datatype data_variable2;
    ......................

    // Method definitions
    modifier method1(datatype parameter11, datatype parameter12, ..) {
        //method definition
    }

    modifier method2(datatype parameter21, datatype parameter22, ....) {
            //method definition
        }......................
        ......................
}
```

Structure of Java application (continued)

Class Example

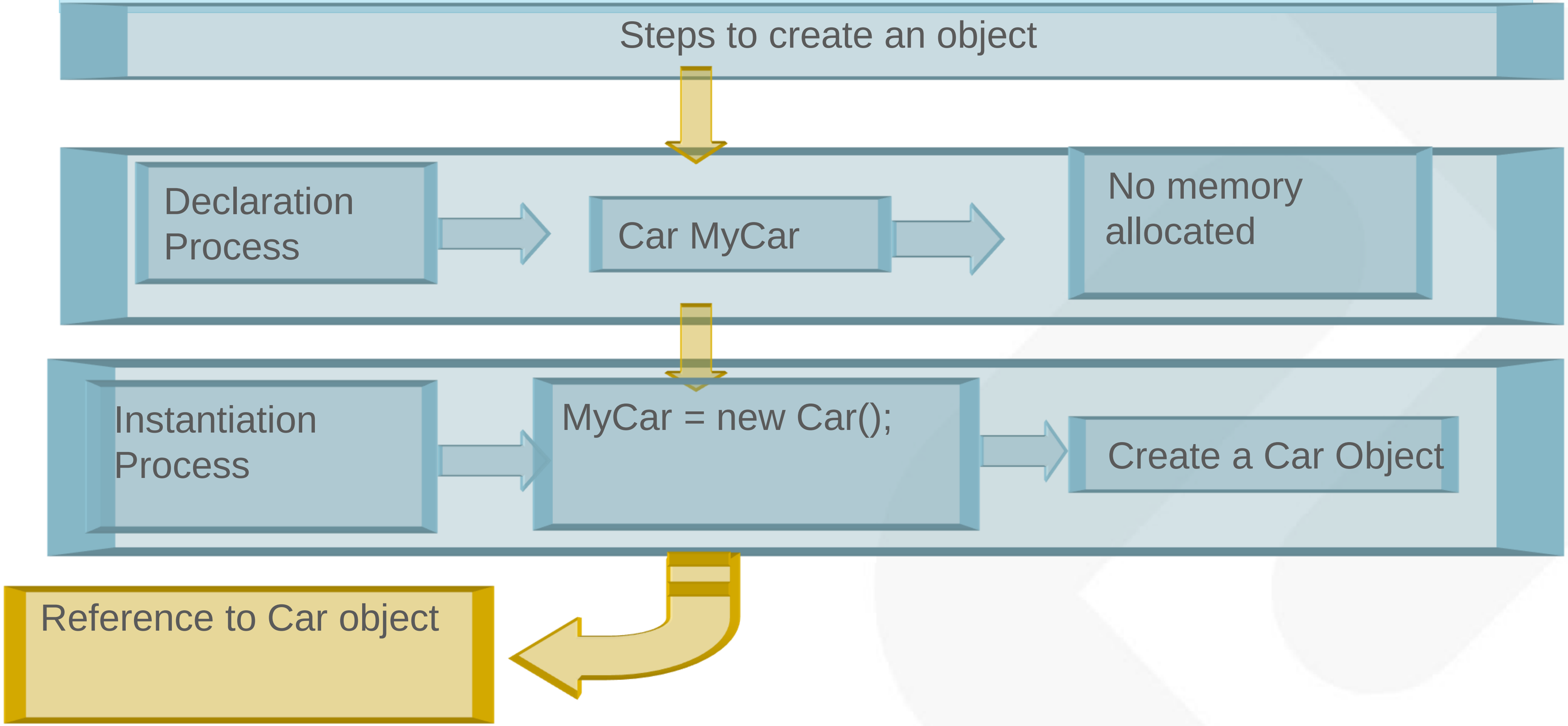


```
class Car {
    String brand;
    String color;
    int wheels;
    int speed;
    int gear;

    void accelerate() {
    }

    void brake() {
    }

    void changeGear() {
    }
}
```

Structure of Java application (continued)

Object of Classes

- An object is an instance of a class having a unique identity.



Steps to create an object

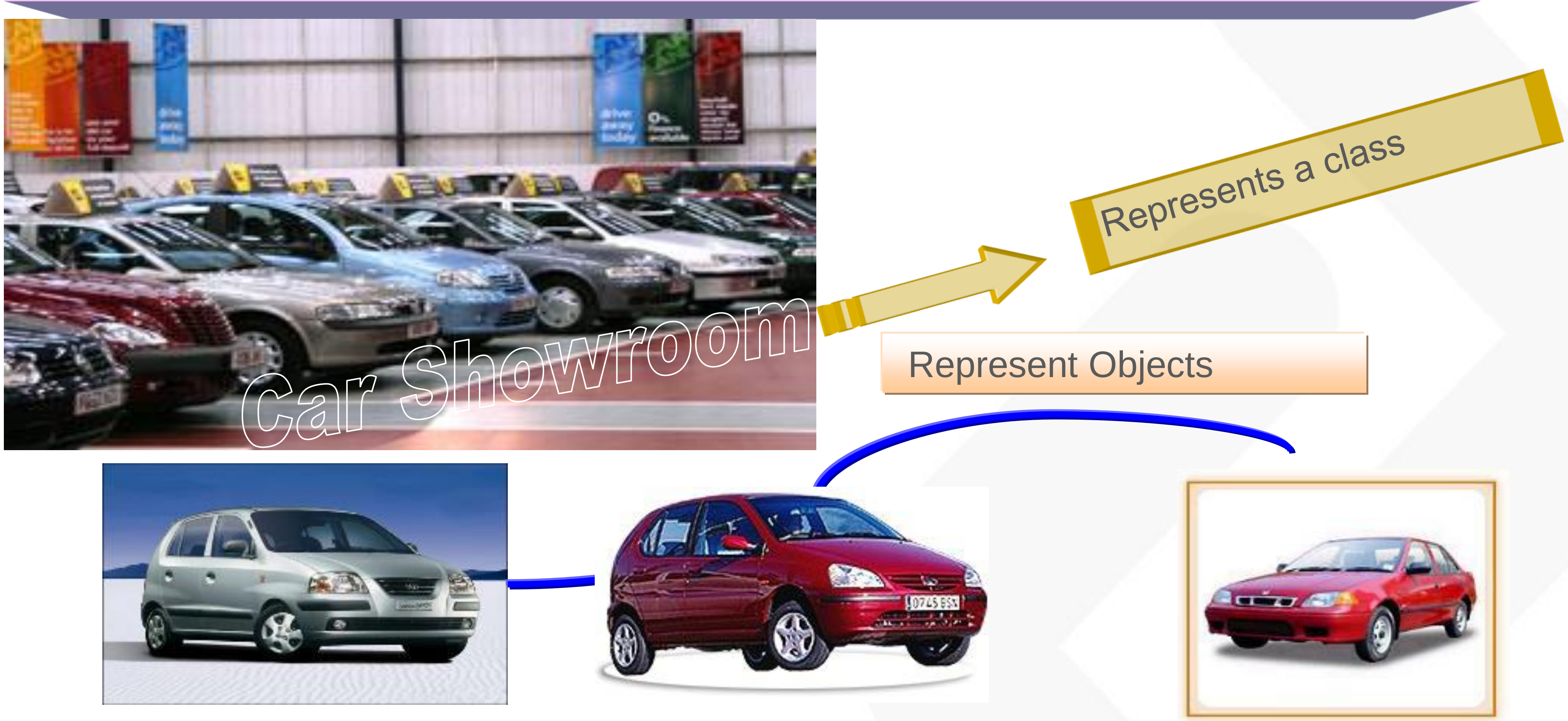
Declaration Process → Car MyCar → No memory allocated

Instantiation Process → MyCar = new Car(); → Create a Car Object

Reference to Car object

Structure of Java application (continued)

Object of Classes (Continued)



Represents a class

Represent Objects

Structure of Java application (continued)

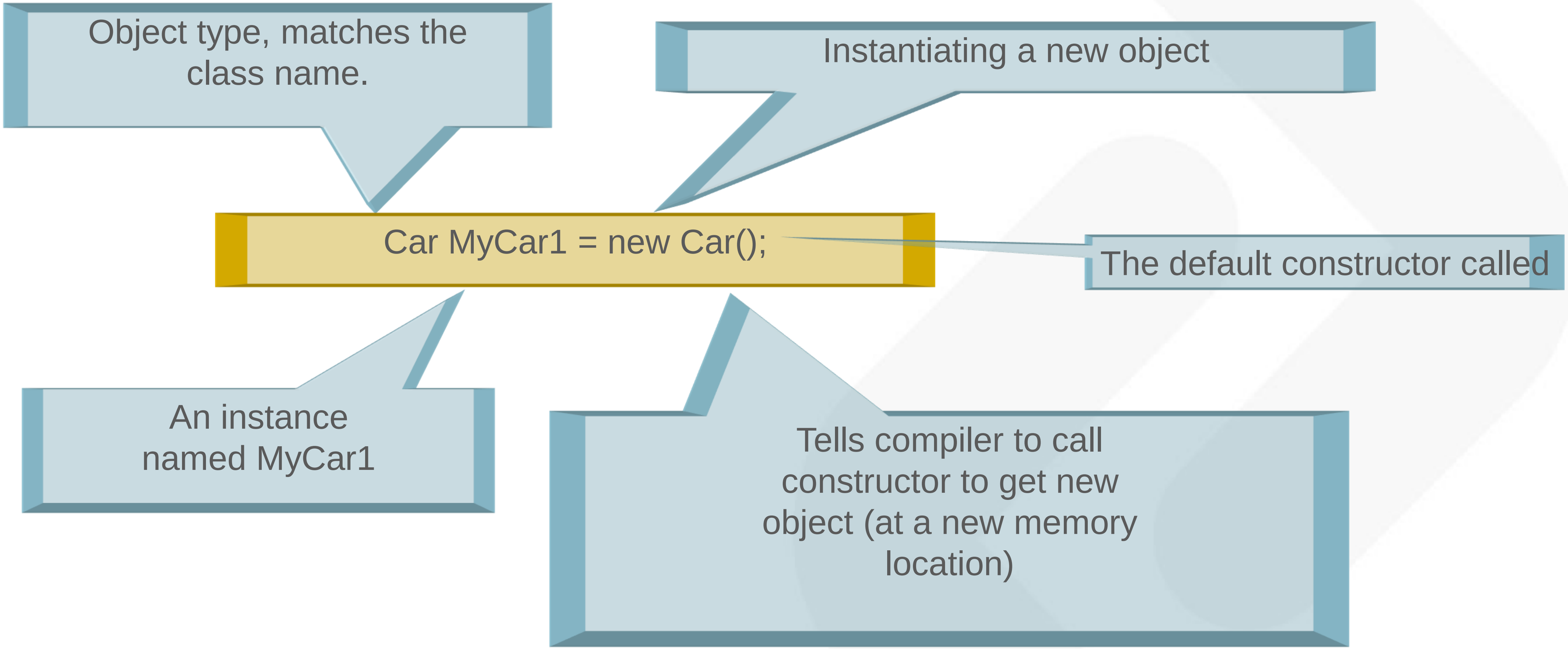
Object of Classes (Continued)

Represents a class

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

Structure of Java application (continued)

Object of Classes (Continued)



Car MyCar1 = new Car();

Object type, matches the class name.

Instantiating a new object

The default constructor called

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

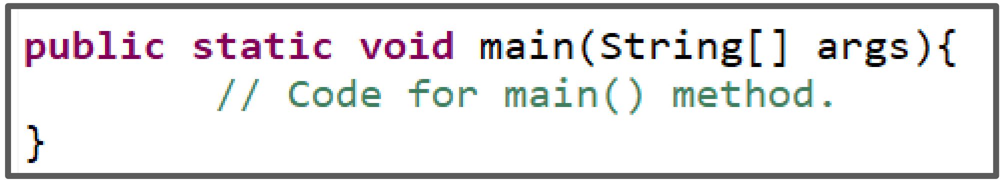
- Dot Operator(.) is used to access the data members of a class outside the class by specifying the data member name or the method name.

```
MyCar.brand = “Maruti Esteem";
MyCar.color = “Red”;
```

Structure of Java application (continued)

Main Method For Java Application

Syntax of main method:



```
public static void main(String[] args){
         // Code for main() method.
}
```

: public specifies, method can be accessed from any object in a Java program.

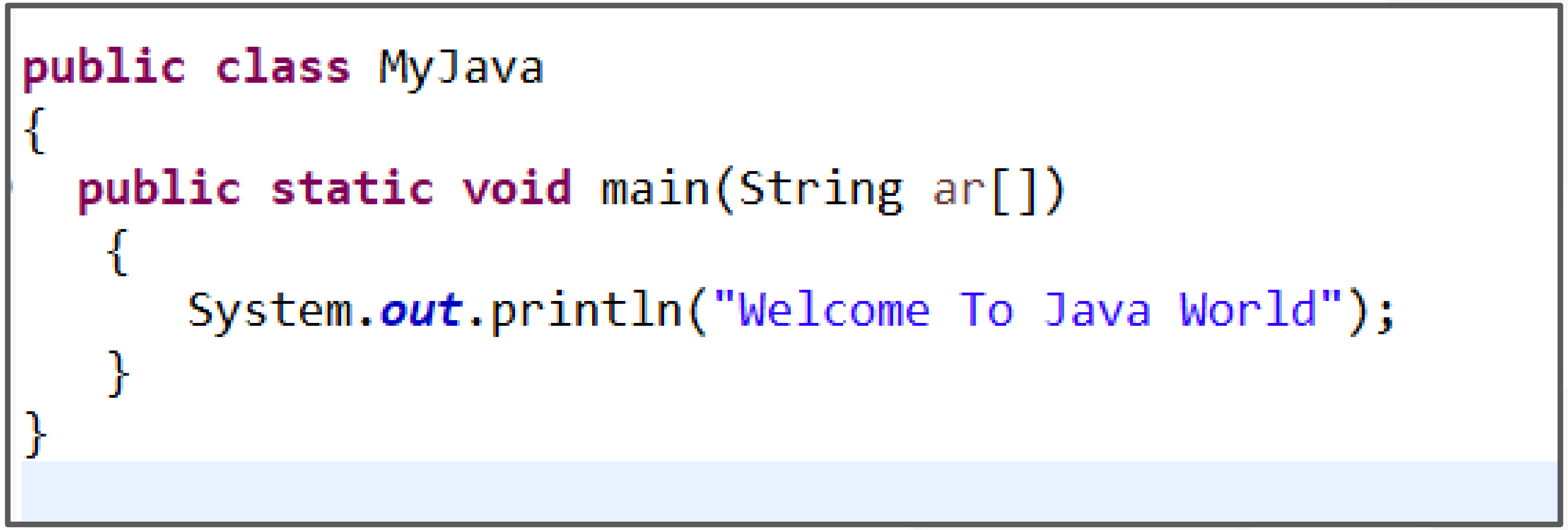
: static keyword associates the method with its class.

: void is the return type of the method.

: main is the name of the method and accepts a single argument in the
form of a String array, which accepts command line arguments.

Structure of Java application (continued)

Sample Java Application



```
public class MyJava
{
  public static void main(String ar[])
   {
      System.out.println("Welcome To Java World");
   }
}
```

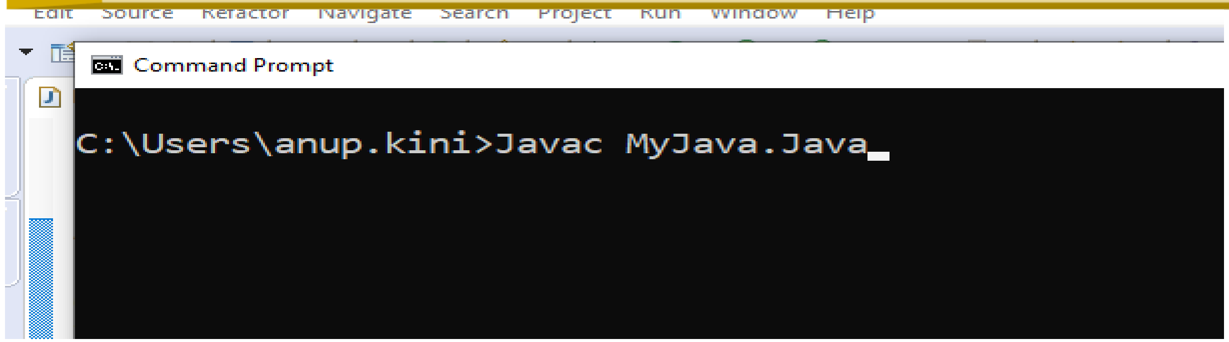
Structure of Java application (continued)

Compiling Java Application

- Save the file that is exactly same as the class name with a .java extension.
  Assume that file is saved in C drive by the name MyJava.Java

- Steps to Compile MyJava.Java application from DOS command prompt

1. Set the path variable to the bin directory of the installation folder of Java.
2. Open the DOS command prompt and type the following command:


Edit Source Refactor Navigate Search Project Run Window Help
Command Prompt

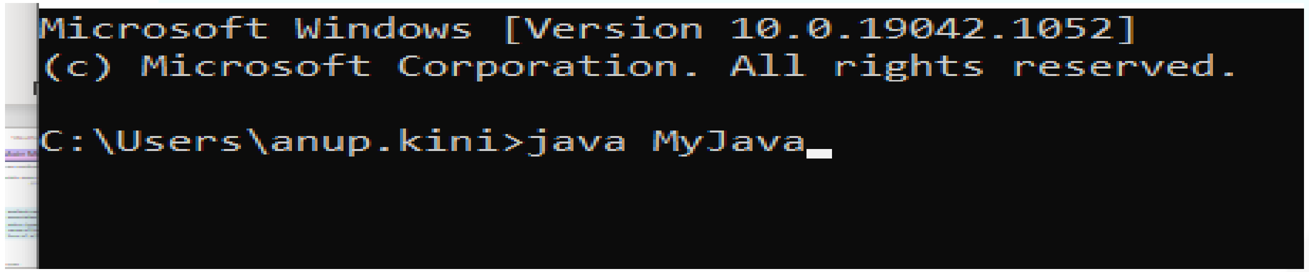
```
C:\Users\anup.kini>Javac MyJava.Java_
```

Structure of Java application (continued)

Run the Application

C:\>

- Use Java command followed by .class file name without using the extension to run the Java application as shown below:



```
Microsoft Windows [Version 10.0.19042.1052]
(c) Microsoft Corporation. All rights reserved.

C:\Users\anup.kini>java MyJava_
```

Press Enter

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 always returns objects of the class type hence there is no return type specified in its definition.
- 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.

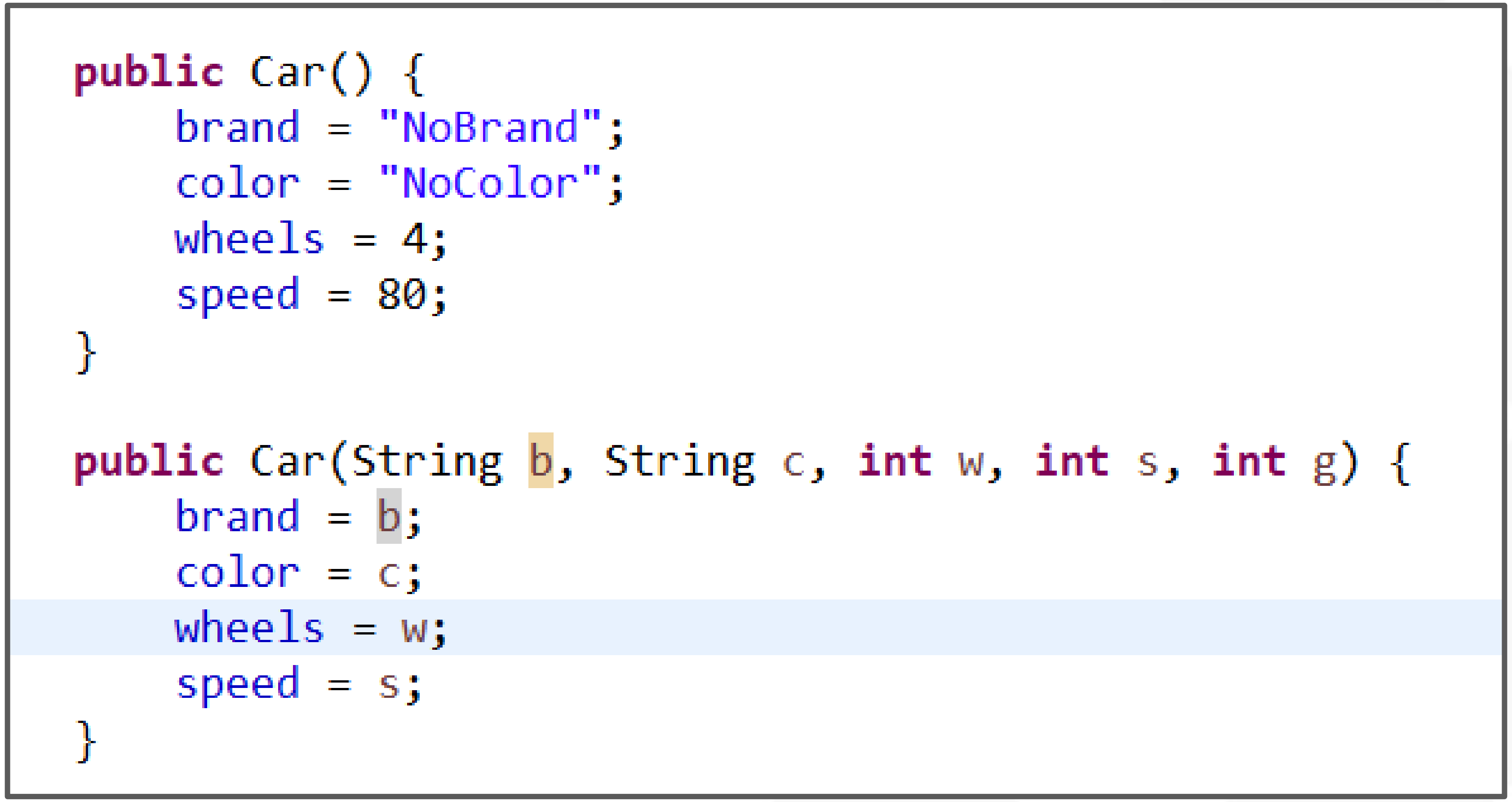
Constructor (continued)

- 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.

Constructor (continued)

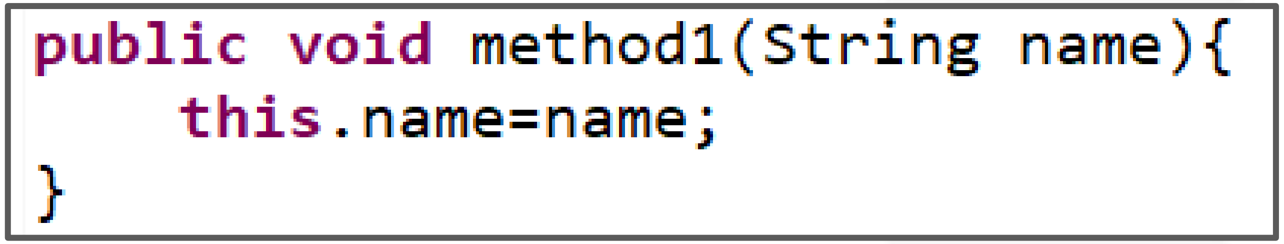


```
public Car() {
    brand = "NoBrand";
    color = "NoColor";
    wheels = 4;
    speed = 80;
}

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

“this” reference

- “this” is a reference to the current object.
- this is used to
  - refer to the current object when it must be passed as a parameter to a method.

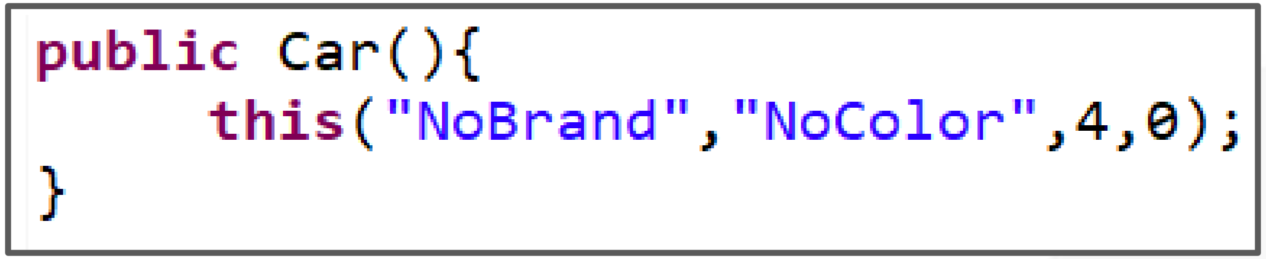
    otherobj1.anymethod(this);
  - refer to the current object when it must be returned in a method.
  - refer the instance variables if parameter names are same as instance variables to avoid ambiguity.



```
public void method1(String name){
    this.name=name;
}
```

“this” reference (continued)

- This keyword included with parenthesis i.e., this() with or without parameters is used to call another constructer. The default construct for the Car class can be redefined as below



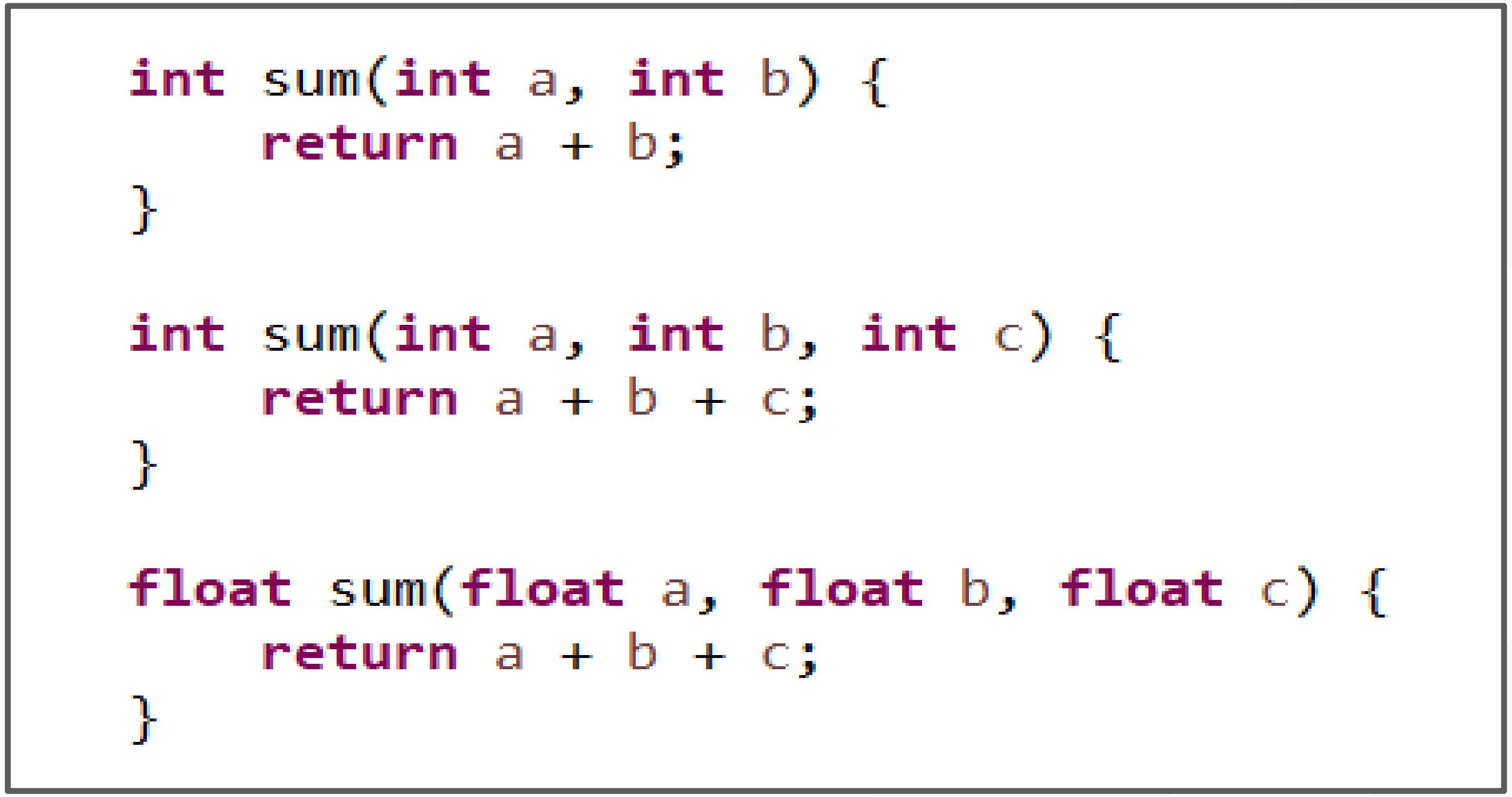
```
public Car(){
     this("NoBrand","NoColor",4,0);
}
```

- The this() can be called only from a constructor and must be the first statement in the constructor

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.

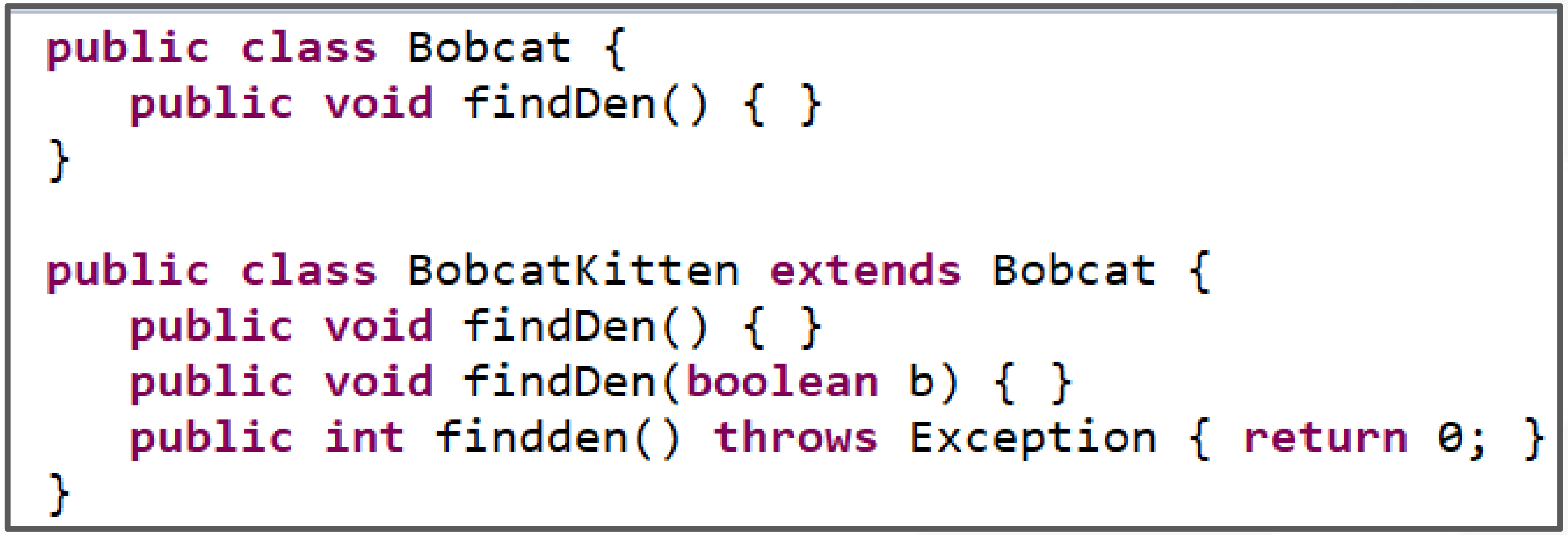
Method overloading : Example



```
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;
}
```

Question on Overloading and Overriding

- Which method(s) in BobcatKitten overload or override the one in Bobcat?



```
public class Bobcat {
   public void findDen() { }
}

public class BobcatKitten extends Bobcat {
   public void findDen() { }
   public void findDen(boolean b) { }
   public int findden() throws Exception { return 0; }
}
```

Answer of Overloading and Overriding

- The one on line 2 is an override because it has the same method signature.

- The one on line 3 is an overloaded method because it has the same method name but a different parameter list.

- The one on line 4 is not an override or overload because it has a different method name. Remember that Java is case sensitive.

Rules of Method Overloading

- **When multiple overloaded methods are present, Java looks for the closest match first. It tries to find the following:**
  - Exact match by type
  - Matching a superclass type
  - Converting to a larger primitive type
  - Converting to an autoboxed type
  - Varargs

Rules of Method Overriding

- **For Overriding, the overridden method has a few rules:**
  - The access modifier must be the same or more accessible.
  - The return type must be the same or a more restrictive type, also known as covariant return types.
  - If any checked exceptions are thrown, only the same exceptions or subclasses of those exceptions are allowed to be thrown.
  - The methods must not be static. (If they are, the method is hidden and not overridden.)

Method Overloading vs Overriding

- Overloading and Overriding happen only when the method name is the same.
- Overriding occurs only when the method signature is the same.
- The method signature is the method name and the parameter list.
- For overloading, the method parameters must vary by type and/or number.

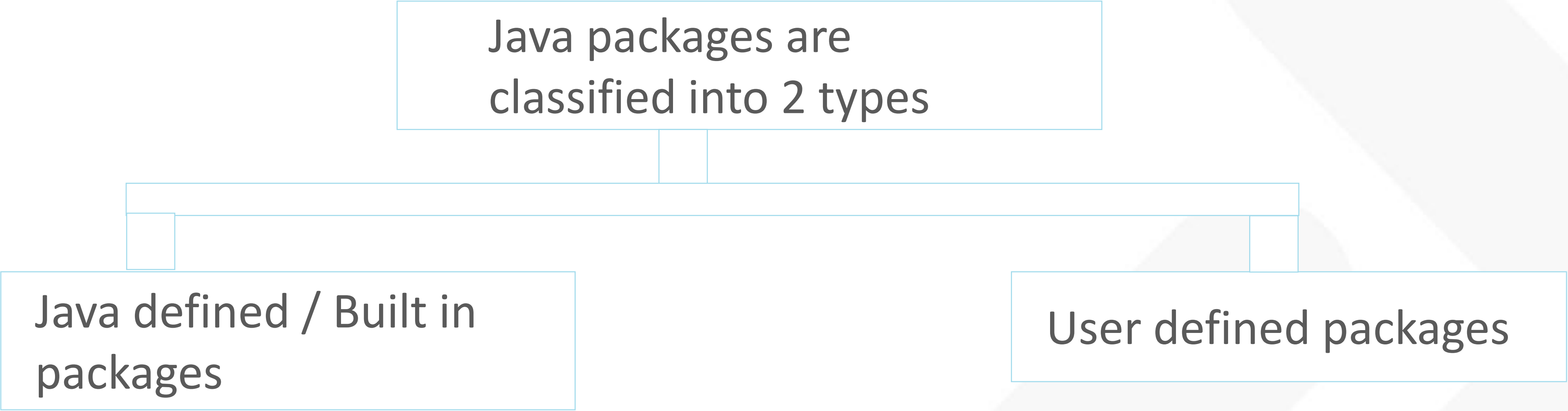
Understanding usage of packages

- "A package is a namespace that organizes 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.

- 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.

Types of packages



Java packages are classified into 2 types

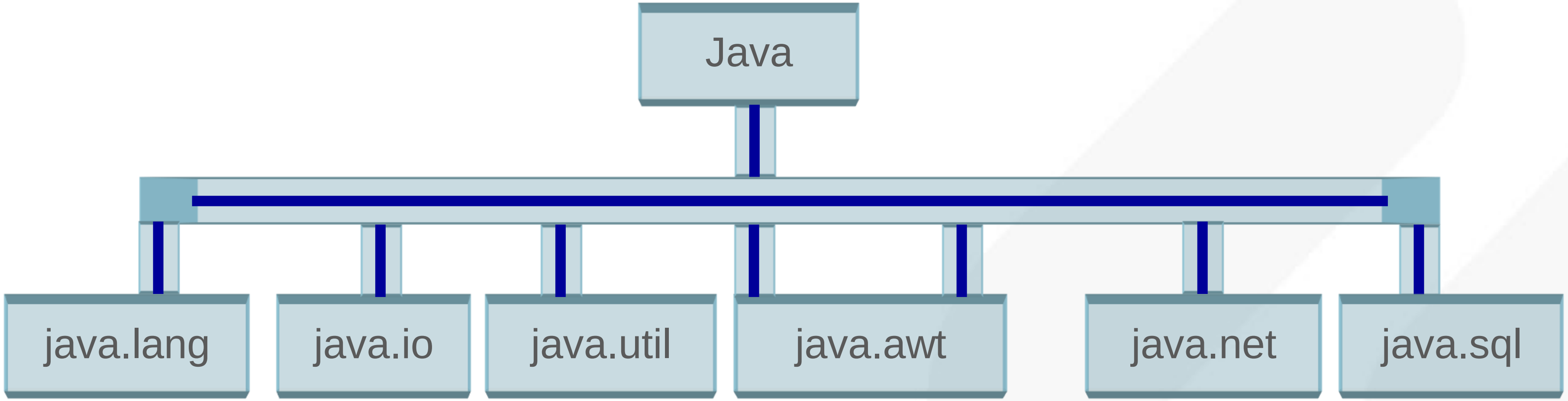
Java defined / Built in packages

User defined packages

Packages in Java

Hierarchy of Java Packages

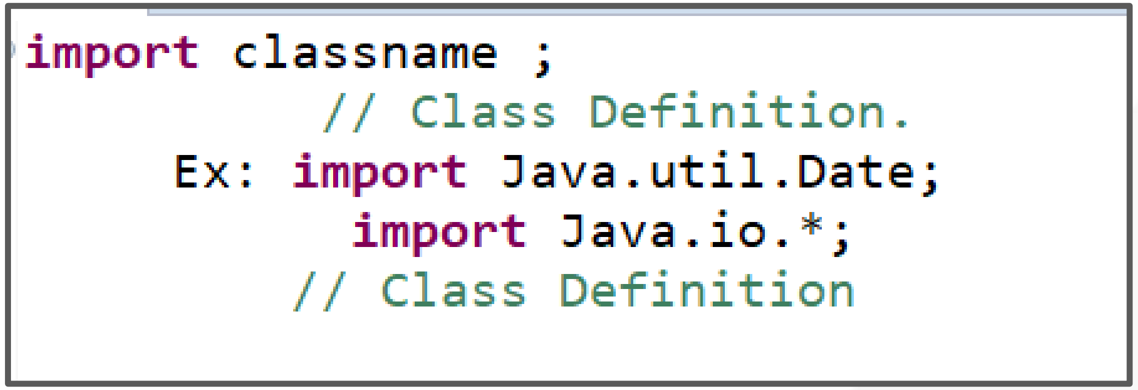
The packages in Java can be nested. The ’ Java ‘ is the root package for all the packages of Java API. Below is the partial hierarchy of Java API.



The classes of a package are qualified with package name

Significance of import keyword

- The classes in the packages are to be included into the applications as required.
- The classes of a package are included using import keyword
- The import keyword followed by fully qualified class name is to be placed before the application class definition.



```
import classname ;
            // Class Definition.
      Ex: import Java.util.Date;
              import Java.io.*;
           // Class Definition
```

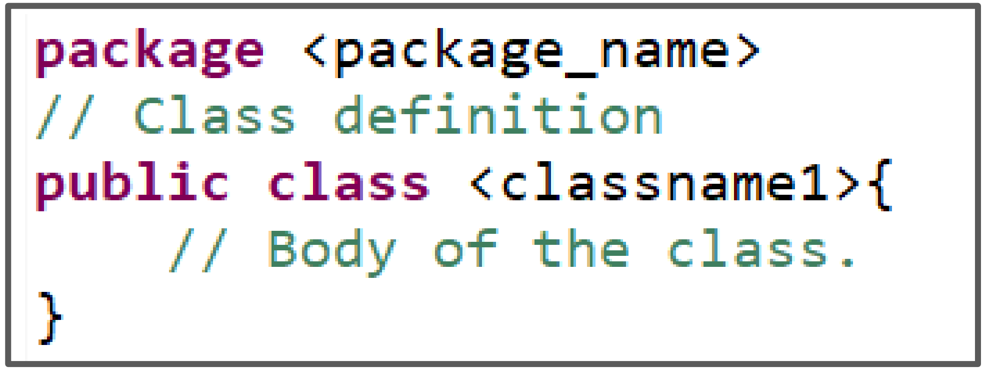
- above indicates to import all the classes in a package.
- All the classes in Java.lang package are included implicitly.

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.

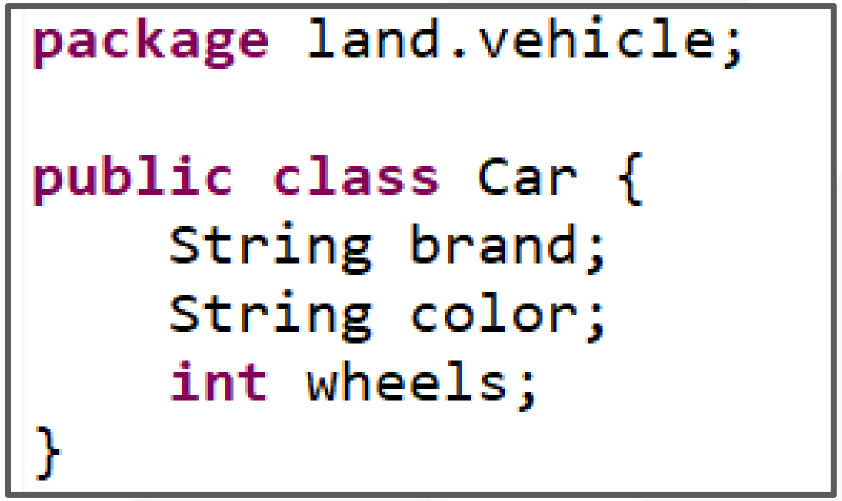
User defined packages (continued)

Syntax for Creating Package



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

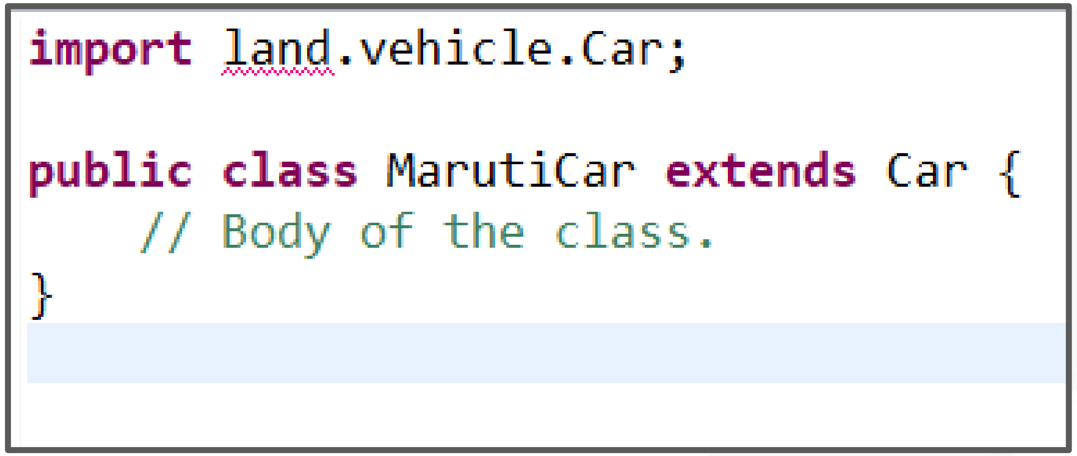
Example



```
package land.vehicle;

public class Car {
    String brand;
    String color;
    int wheels;
}
```

User defined packages (continued)

- 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



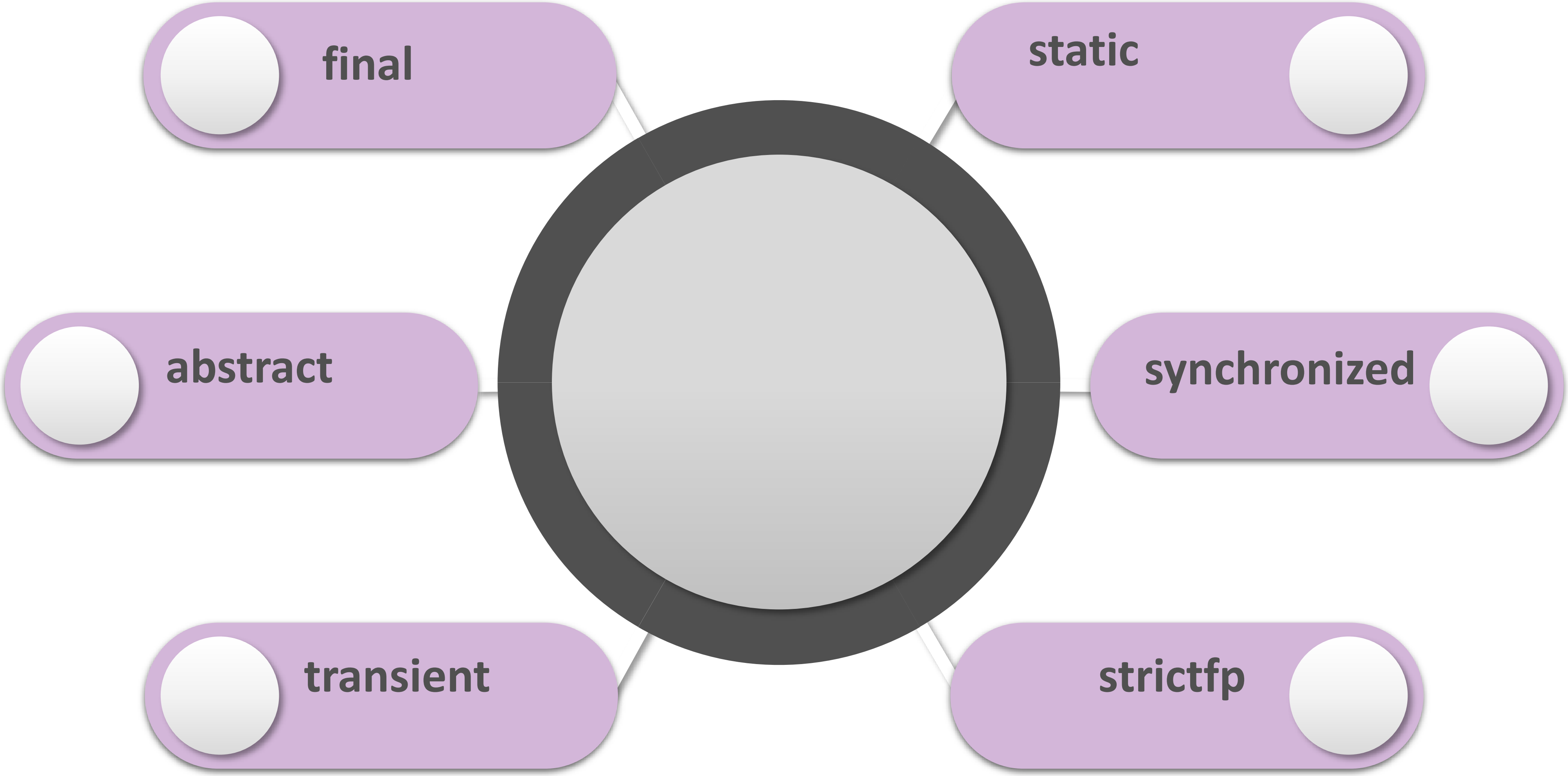
```
import land.vehicle.Car;

public class MarutiCar extends Car {
    // Body of the class.
}
```

Access modifiers

Can access	If that member is private?	If that member has default (package private) access?	If that member is protected?	If that member is public?
Member in the same class	yes	yes	yes	yes
Member in another class in the same package	no	yes	yes	yes
Member in a superclass in a different package	no	no	yes	yes
Method/field in a class (that is not a superclass) in a different package	no	no	no	yes

Modifiers

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



Types of modifiers

- **static keyword**

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

- Methods and data members can be called without instantiating the class.

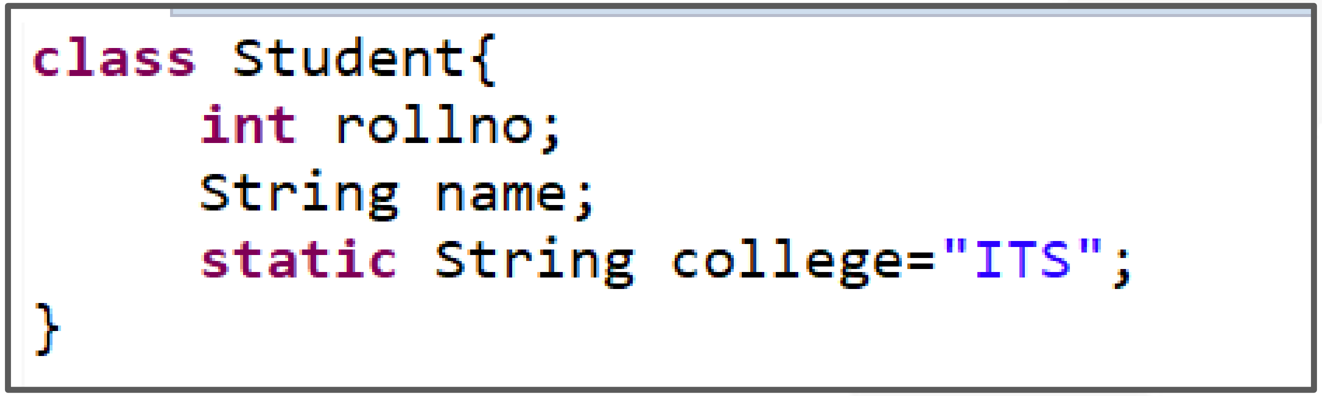
- A static member exists when no objects of that class exist.

static modifier

- The static can be:
  - Variable (also known as a class variable)
  - Method (also known as a class method)
  - Block
  - Nested class

static variable

- If you declare any variable as static, it is known as a static variable.
- The static variable can be used to refer to the common property of all objects (which is not unique for each object), for example, the company name of employees, college name of students, etc.
- The static variable gets memory only once in the class area at the time of class loading.

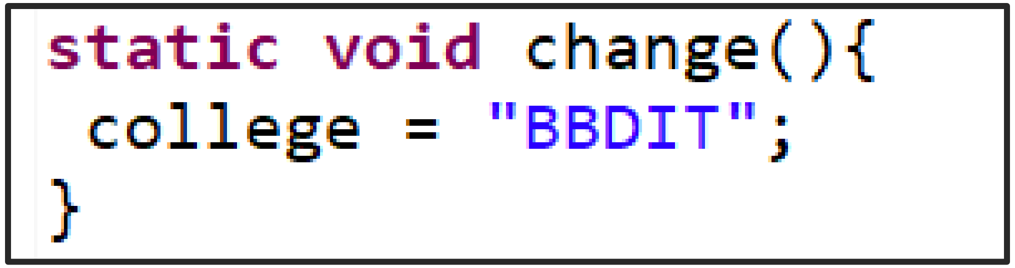


```
class Student{
     int rollno;
     String name;
     static String college="ITS";
}
```

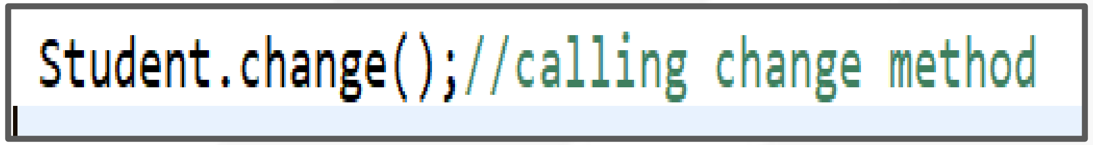
static method

- If you apply static keyword with any method, it is known as static method.

- A static method belongs to the class rather than the object of a class.
- A static method can be invoked without the need for creating an instance of a class.
- A static method can access static data member and can change the value of it.



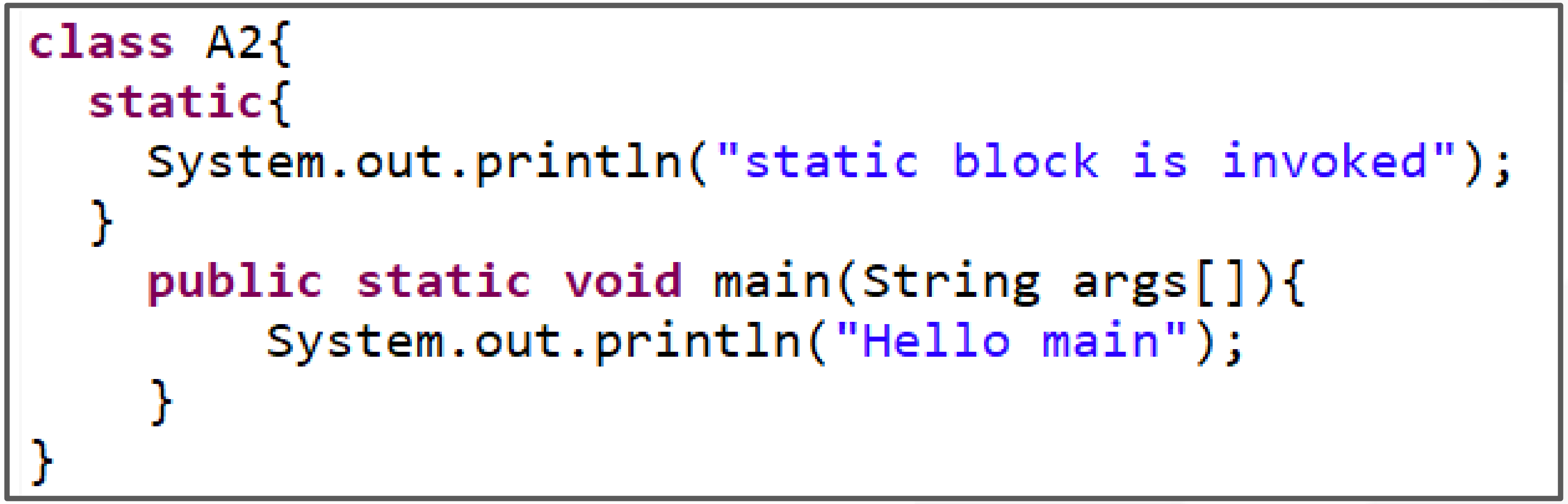
```
static void change(){
 college = "BBDIT";
}
```



```
Student.change();//calling change method
```

static block

- Is used to initialize the static data member.
- It is executed before the main method at the time of class loading.



```
class A2{
  static{
    System.out.println("static block is invoked");
  }
    public static void main(String args[]){
        System.out.println("Hello main");
    }
}
```

Static vs. instance calls

Type	Calling	Legal?	How?
Static method	Another static method or variable	Yes	Using the class name
Static method	An instance method or variable	No	
Instance method	A static method or variable	Yes	Using the classname or a reference variable
Instance method	Another instance method or variable	Yes	Using a reference variable

Types of modifiers

- **final keyword**

- A final data members cannot be modified.

- A final method cannot be overridden in the subclass.

- A class declared as final cannot be inherited .

final vs Immutability in Java

- **final :** In Java, final is a modifier which is used for class, method and variable also. When a variable is declared with final keyword, it’s value can’t be modified, essentially, a constant.

- **Immutability** : In simple terms, immutability means unchanging over time or unable to be changed.
- In Java, String objects are immutable means we can't change anything to the existing String objects.

final vs Immutability in Java

- final means that you can’t change the object’s reference to point to another reference or another object, but you can still mutate its state (using setter methods e.g.).

- Whereas immutable means that the object’s actual value can’t be changed, but you can change its reference to another one.

- final modifier is applicable for variable but not for objects, Whereas immutability applicable for an object but not for variables.

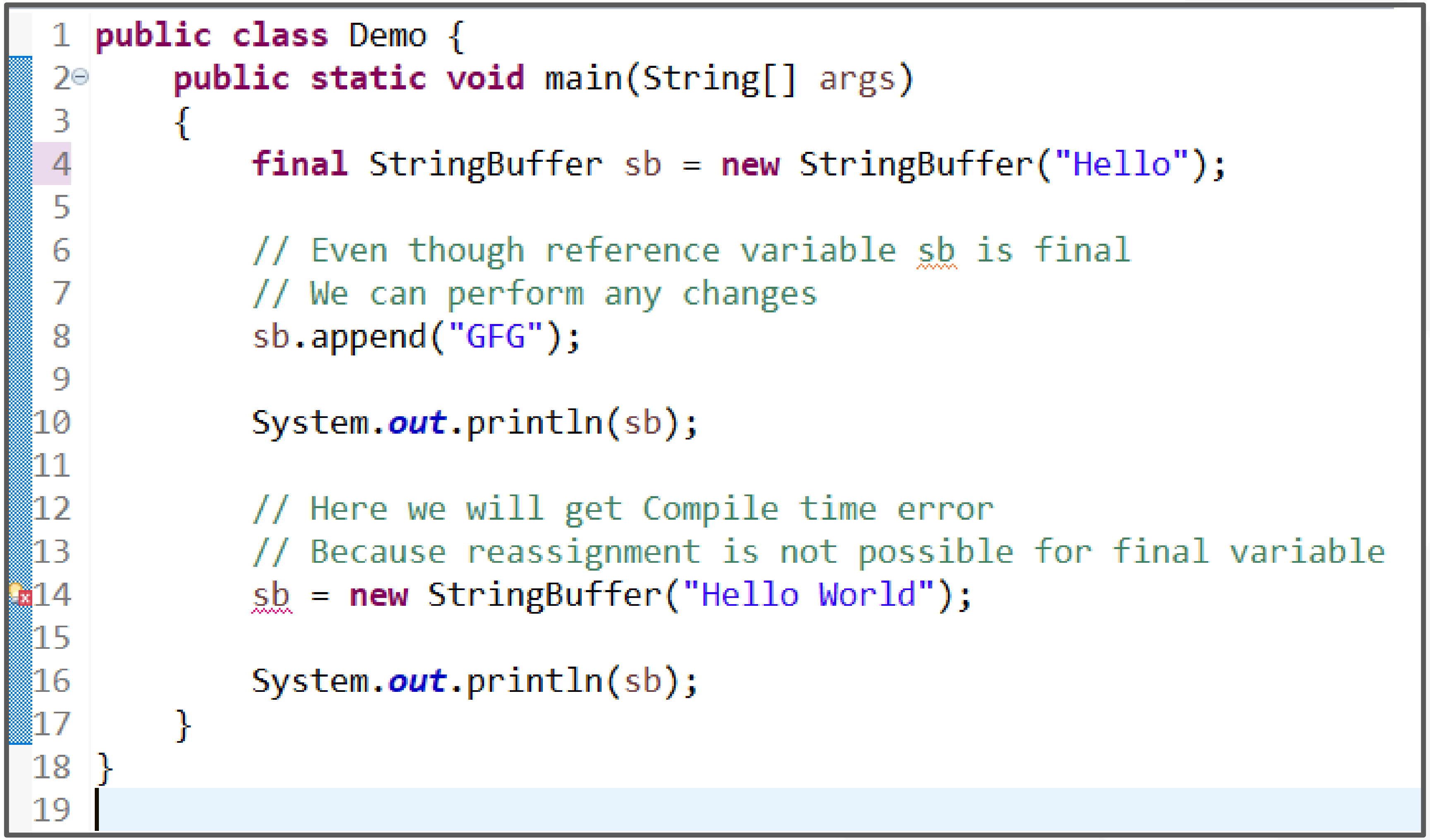
final vs Immutability in Java

- By declaring a reference variable as final, we won’t get any immutability nature, Even though reference variable is final.

- We can perform any type of change in the corresponding Object. But we can't perform reassignment for that variable.

- final ensures that the address of the object remains the same whereas the Immutable suggests that we can’t change the state of the object once created.

final vs Immutability in Java



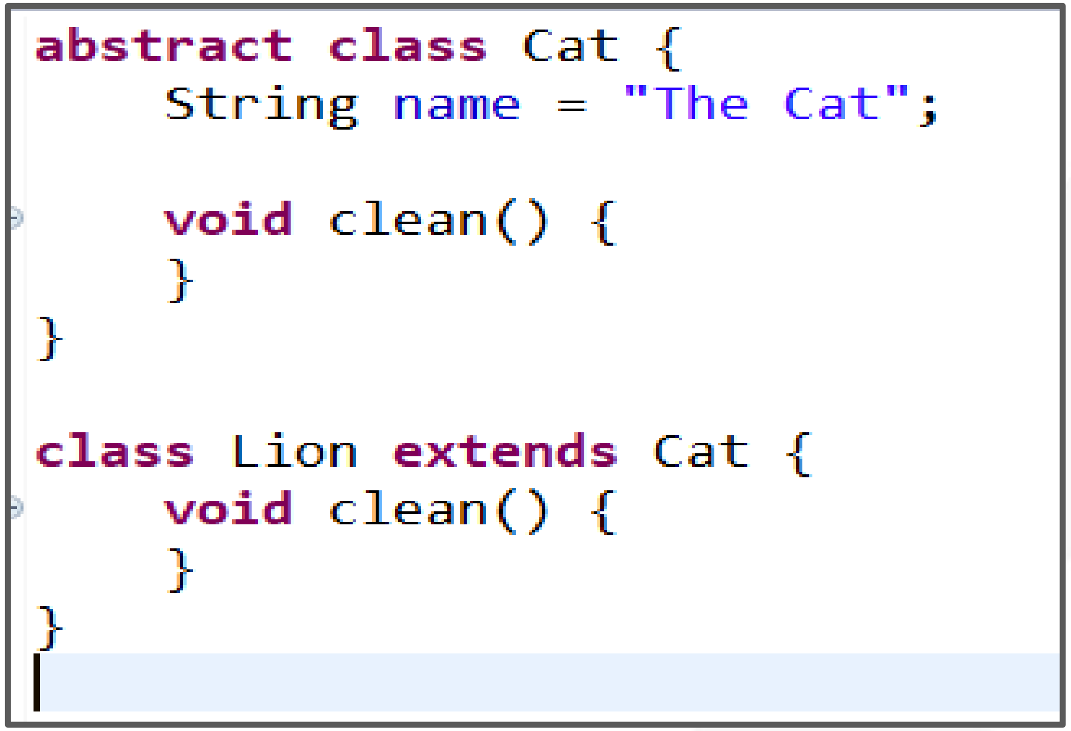
```java
public class Demo {
    public static void main(String[] args)
    {
        final StringBuffer sb = new StringBuffer("Hello");

        // Even though reference variable sb is final
        // We can perform any changes
        sb.append("GFG");

        System.out.println(sb);

        // Here we will get Compile time error
        // Because reassignment is not possible for final variable
        sb = new StringBuffer("Hello World");

        System.out.println(sb);
    }
}

```

Note on static and final modifiers

- **final** prevents a variable from changing or a method from being overridden.
- **static** makes a variable shared at the class level and uses the class name to refer to a method.
- **static** and **final** are allowed to be added on the class level too.
- Using **final** on a class means that it cannot be subclassed.
- As with methods, a class cannot be both **abstract** and **final**. In the Java core classes, **String** is final.

Question on static and final modifier

- **To which lines in the following code could you independently add static and/or final without introducing a compiler error?**



```
abstract class Cat {
    String name = "The Cat";

    void clean() {
    }
}

class Lion extends Cat {
    void clean() {
    }
}
```

Answer of static and final modifier

- Both static and final can be added to line 2. This allows the variable to be accessed as Cat.name and prevents it from being changed.

- static cannot be added to line 3 or 6 independently because the subclass overrides it. It could be added to both, but then you wouldn't be inheriting the method.

- The final keyword cannot be added to line 3 because the subclass method would no longer be able to override it.

- final can be added to line 6 since there are no subclasses of Lion.

Types of modifiers

- **abstract keyword**

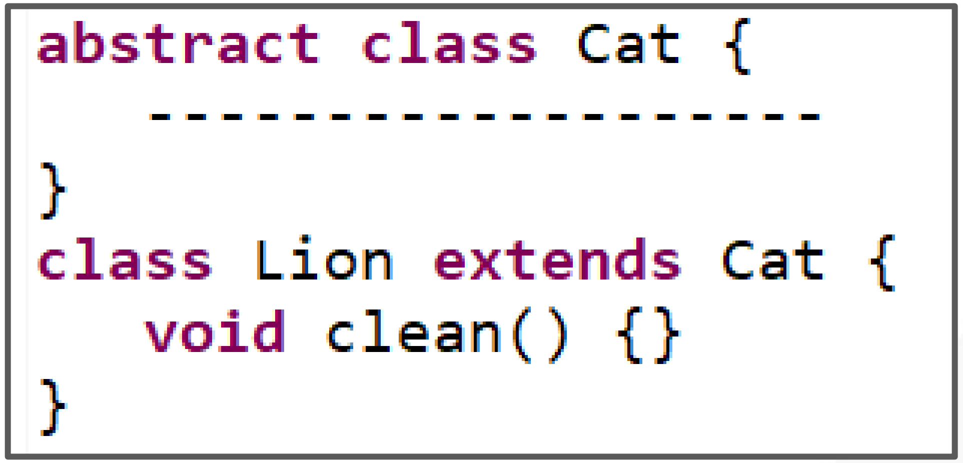
- Used to declare class that provides common behavior across a set of subclasses.

- Abstract class provides a common root for a group of classes.
- Abstract methods may not appear in a class that is not abstract. The first concrete subclass of an abstract class is required to implement all abstract methods that were not implemented by a superclass.

- An abstract keyword with a method does not have any definition.
- An abstract class may contain any number of methods including zero. The methods can be abstract or concrete.

Question on abstract modifier

- **What are three ways that you can fill in the blank to make this code compile?**



```
abstract class Cat {
   -------------------
}
class Lion extends Cat {
   void clean() {}
}
```

Answer of abstract modifier

- One of them is a little tricky. The tricky one is that you could leave it blank. An abstract class is not required to have any methods in it, let alone any abstract ones.

- A second answer is the one that you probably thought of right away:

```
        abstract void clean();
```

- A third answer is a default implementation:

```
        void clean () {}
```

Types of modifiers

synchronized keyword

- Controls the access to a block in a multithreaded environment.

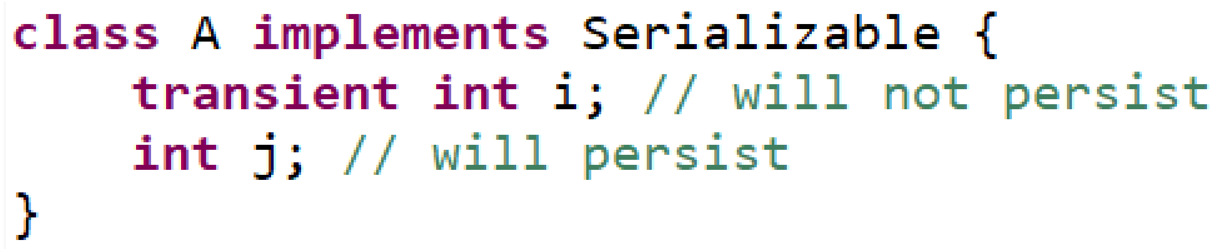
- Synchronized keyword is used as a block.

- Synchronized keyword is used in a method definition.

Types of modifiers

transient keyword

- A transient variable is a variable that may not be serialized JVM understands that the indicated variable is not part of the persistent state of the object



```
class A implements Serializable {
    transient int i; // will not persist
    int j; // will persist
}
```

Here, if an object of type A is written to a persistent storage area,

the contents of i would not be saved, but the contents of j would.

Types of modifiers

strictfp keyword

- strictfp can be used to modify a class or a method, but never a variable.
- Marking a class as strictfp means that any method code in the class will conform to the IEEE 754 standard rules for floating points.

- Without that modifier floating points used in the methods might behave in a platform-dependent way.

- If not at class level, strictfp behavior can be applied at method level, by declaring a method as strictfp.
- Note: Usage of strictfp is required when floating point values should be consistent across multiple platforms

Quick Summary

- An object is a software package consisting of variables and methods.
- Various programming methodologies that can be used are:
  - Procedural programming.
  - Object-oriented programming.
- The procedural programming methodology involves dividing a large program into a set of sub procedures or subprograms that perform specific tasks.
- The procedural programming methodology allows code reusability in large applications.

Quick Summary

- An object is defined as an instance of a class.
- In the object-oriented approach, classes are designed such that they can be reused.
- Object Oriented programming offers features such as Reusability, Resilience, Object oriented Modularity, and Information hiding.
- Encapsulation- Hides the implementation details of an object and therefore hides its complexity.
- Abstraction- Focuses on the essential features of an object.

Quick Summary

- Inheritance - Creates a hierarchy of classes and helps in reuse of attributes and methods of a class.
- A superclass shares its attributes and behavior with its child classes.
- There are two types of inheritance:
- Single inheritance- A class inherits attributes from only one superclass.
- Multiple inheritance- A class inherits attributes from two or more
- super classes.
- Polymorphism- Assigns a different meaning or usage to an entity in different contexts.

Review questions

1. _ _ _ _ _ _ programming involves a large program into a set of sub procedures or subprograms that perform specific tasks.

   a. Sequential

   b. Procedural

   c. Object Oriented

   d. Rule-Based

2. _ _ _ _ _ _ among the following operates on the data.

   a. Variables

   b. Class

   c. Method

   d. All the above

Review questions

3. What is the acronym for OOP?

    a. Object Oriented Programming

    b. Object Orientation programming

    c. Object Oriented Program

    d. Object Oriented Procedure

4. _ _ _ _ _ _ _ is the concept that an object should totally separate its interface from its implementation

    a. Abstraction

    b. Polymorphism

    c. Encapsulation

    d. Inheritance

Review questions

5. A user-defined data type that defines a collection of objects that share the same characteristics.

   a.Object

   b.Class

   c.Array

   d.Method

6. Identify the first step in doing Object Oriented Programming.

   a.Data Designing

   b.Data Modeling

   c.Class Modeling

   d.Object Design

Review questions

7. _______ the ability of a generalized request (message) to produce different results based on the object that it is sent to.

a. Inheritance

b. Abstraction

c. Polymorphism

d. Encapsulation

8. Identify the software construct that encapsulates data, along with the ability to use or modify that data, into a software entity.

a. Object

b. Class

c. Method

d. Variable

Review questions

9. _ _ _ _ _ _ feature of OOPS that means ignoring the non-essential details of an object and concentrating on its essential features?

    a.Inheritance
    b.Encapsulation
    c.Abstraction
    d.Polymorphism

10.Pick the odd man out.

    a. State
    b. Class
    c. Behavior
    d. Identity

Recap

- Define Class, Methods & Properties
- Working with constructors
- Use of “this” keyword
- Method overloading
- Understand usage of Packages
- Creating user defined packages
- Managing classes under packages
- Using package members
- Access modifiers
- Significance of import keyword
- Different modifiers (final, static, abstract etc.)

Mphasis
The Next Applied

THANK YOU

Important Confidentiality Notice

This document is the property of, and is proprietary to Mphasis, and identified as “Confidential”. Those parties to whom it is distributed shall exercise the same degree of custody and care afforded their own such information. It is not to be disclosed, in whole or in part to any third parties, without the express written authorization of Mphasis. It is not to be duplicated or used, in whole or in part, for any purpose other than the evaluation of, and response to, Mphasis’ proposal or bid, or the performance and execution of a contract awarded to Mphasis. This document will be returned to Mphasis upon request.