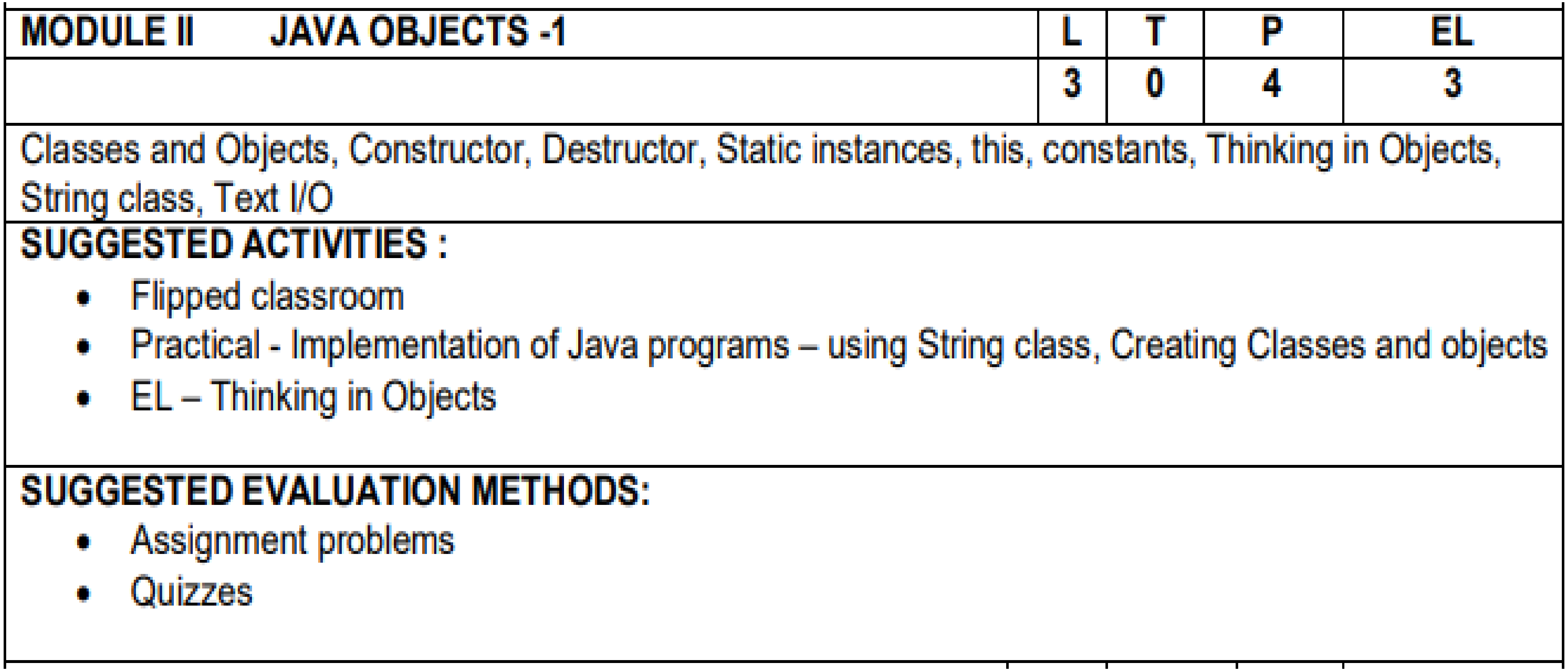
CS6308- Java Programming

V P Jayachitra
Assistant Professor
Department of Computer Technology
MIT Campus
Anna University



MODULE II JAVA OBJECTS -1	L	T	P	EL
	3	0	4	3

Classes and Objects, Constructor, Destructor, Static instances, this, constants, Thinking in Objects, String class, Text I/O

SUGGESTED ACTIVITIES :

- Flipped classroom
- Practical - Implementation of Java programs – using String class, Creating Classes and objects
- EL – Thinking in Objects

SUGGESTED EVALUATION METHODS:

- Assignment problems
- Quizzes

Introduction

- **procedural programming:** Programs that perform their behavior as a series of steps to be carried out.
- **object-oriented programming (OOP)**: Programs that perform their behavior as interactions between objects

About ?

- The basic elements of a class

- How a class can be used to create objects?

- Learn about methods, constructors, and the **this** keyword.

Class

- Any concept to implement in a Java program must be encapsulated within a class.
- What is a class?
  - A class is a structure that defines the data and the methods to work on that data.
  - A new data type
  - A class is an encapsulation of attributes and methods.
  - A class is a logical construct or template.

Class Fundamentals

- Class
  - Defines a new data type
  - A class is to create an object or instance of the defined type
  - A class is a template for an object
  - A class is the blueprint of an object
  - A class describe object’s properties and behaviors
- Example: Blueprint of a house (class) and the house (object)

Class

Properties : State/variables

Behaviors: Methods

Blueprint

```
public class Point
{
    int x;
    int y;
}
```

Instance

Point p=new Point();

Class Clock

Properties : State/variables

int Hour, minute, second;

Behaviors: Methods

int getMinutes();

int getSeconds();

int getHours();

Container class vs Definition class

- Container class:
  - Collection of static methods that are not bound to a specific objects.
  - Example: Math.sqrt(), Math.pow()
- Definition class:
  - A class that create new objects.
  - Example: Clock c1;

Object

- Object
  - An object is an instance of a class.
  - An entity that combines state and behavior
  - An object is a real world entity.

Characteristics of an Object:

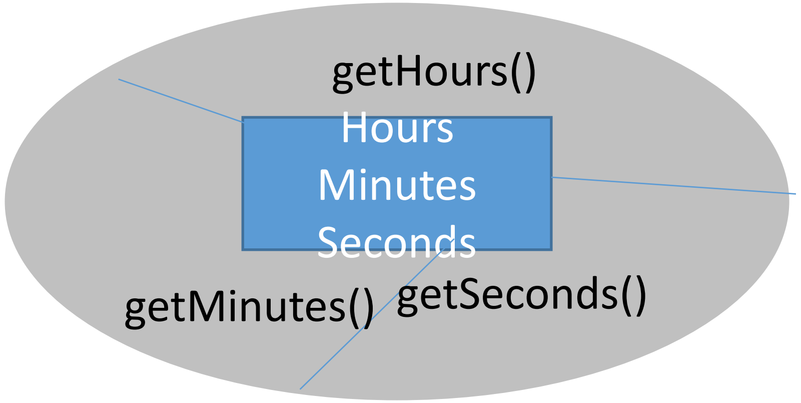
- State : represents the data (value) of an object.
- Behavior : represents the behavior (functionality) of an object.
- Identity : An object identity is typically implemented via unique ID. The value of the ID is not visible to the external user. However, JVM can identify each object uniquely.

Class Abstraction

- Abstract data type (ADT)
  - Abstraction refers to the act of representing essential features without including the background details or explanations.
  - Class is an ADT as it uses the concept of abstraction.
- Abstraction:
  - An abstraction is a process of hiding the implementation details from the user, only the functionality will be provided to the user.
  - Allow user to understand its external behaviour(interface) but not its internal details.
  - Example: Buttons
  - Why? To manage complexity
  - abstraction is achieved by using Interfaces and Abstract classes.

Encapsulation

- State is *encapsulated* or *hidden*
  - The internal state of an object is not directly accessible to other parts of the program
  - Other parts of the program can only access the object using its interface.
  - Data-hiding: Data of a class is hidden from any other class and can be accessed only through any member function of it’s own class in which they are declared.



General form of the class

```
access modifier class ClassName {
    access modifier static type classVariable1; // Class variables)
    // ...
     access modifier type instanceVariable1; // Instance variables
    // ...
     access modifier ClassName() {// Default constructor
        // code   }
    // Parameterized constructor
    access modifier ClassName(type param1, type param2, ...) {
        this.instanceVariable1 = param1;
        this.instanceVariable2 = param2;
        // ...                                   }
    // Method with a return type
    access modifier ReturnType methodName1(ParameterType
param1, ParameterType param2, ...) {
        // Method body
        return returnValue;      }
```

```
// Method without a return type (void)
    access modifier void methodName2(ParameterType
param1, ParameterType param2, ...) {
        // Method body
    }

    // Static method
    access modifier static ReturnType
staticMethodName(ParameterType param1, ParameterType
param2, ...) {
        // Method body
        return returnValue;
    }

    // Main method (entry point)
    access modifier static void main(String[] args) {
        // Code to test the class
    }
}
```

- The data, or variables, defined within a class are called instance variables.
  - Each instance of the class (that is, each object of the class) contains its own copy of these variables.
  - Thus, the data for one object is separate and unique from the data for another.
- The code is contained within .defined within a class are called members of the class.
- Classes have a main method(), if that class is the starting point of the program

A simple class

```
class Car {
String make;
String model;
int year;
}
```

Create a Car object, use a statement like the following:

Car myCar = new Car();

// create a Car object called myCar

Instance variables

A class called Car that defines three instance variables:

make, model and year.

Class name

A class defines a new type of data.

In this case, the new data type is called Car.

class name is used to declare objects of type Car.

A class declaration only creates a template;

It does not create an actual object.

Object

myCar will refer to an instance of Car. Thus, it will have “physical” reality.

Each time on creating an instance of a class means creating an object that contains its own copy of each instance variable defined by the class

Every Car object will contain its own copies of the instance variables make, model, and year.

A simple class

```
class Car {
String make;
String model;
int year;
}
```

To actually create a Car object, use a statement like the following:

```
Car myCar = new Car();
// create a Car object called mycar
```

<u>Dot operator</u>

- Use the dot operator to access both the instance variables and the methods within an object.
- The dot operator links the name of the object with the name of an instance variable or methods.

```
myCar.year = 2024;
//assign year variable of myCar the value 2024
```

```
class Car {
   String make;
   String model;
   int year;
   double fuelEfficiency;
}

// This class declares an object of type Car.
class CarDemo {
   public static void main(String args[]) {
      Car myCar = new Car();
      double range;  //local variable
      // myCar instance variables
      myCar.make = "Tesla ";
      myCar.model = " Roadster ";
      myCar.year = 2024;
      myCar.fuelEfficiency = 0.5;
      // compute range of car (assuming 15-gallon tank)
      range = myCar.fuelEfficiency * 15;
      System.out.println("A " + myCar.year + " " + myCar.make + " " + myCar.model +  " can travel approximately " + range +
" miles on a full tank.");
   }
}
```

```
A 2024 Tesla  Roadster can travel approximately 7.5 miles on a full tank.
```

Save the file that contains this program CarDemo.java, because the main( ) method is in the class called CarDemo, not the class called Car.

Compilation will create two .class files, one for Car and one for CarDemo.

Each class can also be defined in its own file, called Car.java and CaeDemo.java,

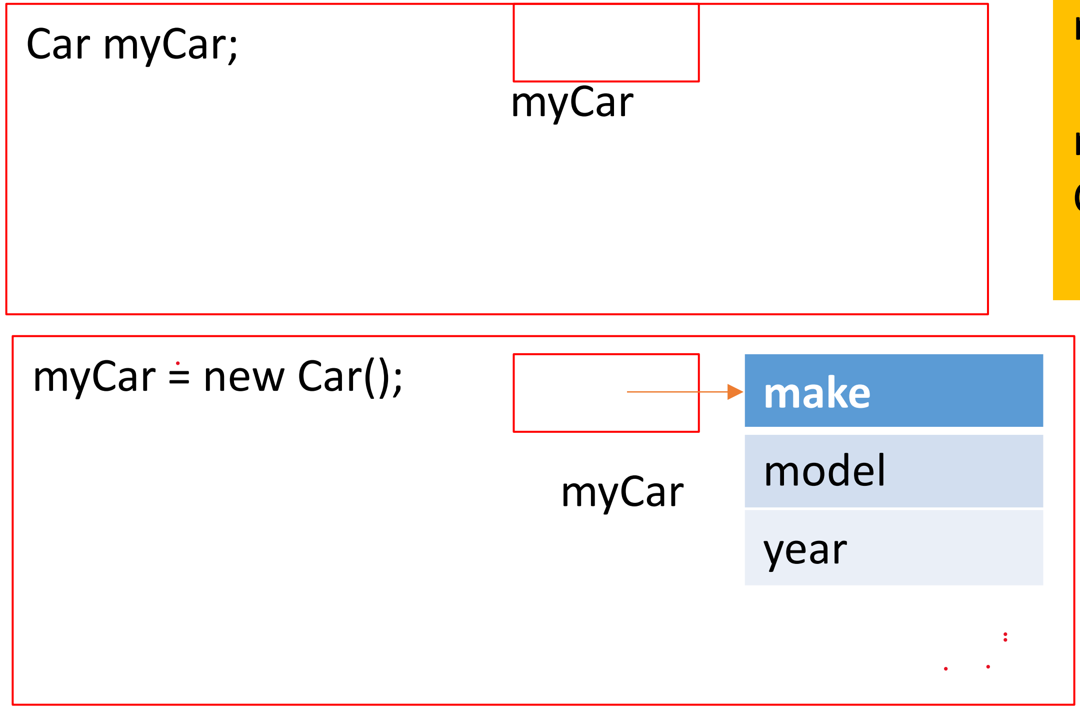
```
//TWO OBJECTS
class Car {
    String make;
    String model;
    int year;
    double fuelEfficiency;  // miles per gallon
}
class CarDemo2 {
    public static void main(String args[]) {
        Car myCar1 = new Car();
        Car myCar2 = new Car();
        double range;

        // assign values to myCar1's instance variables
        myCar1.make = "Tesla";
        myCar1.model = "Roadster";
        myCar1.year = 2024;
        myCar1.fuelEfficiency = 0.5;

        // assign different values to myCar2's instance variables
        myCar2.make = "Tesla";
        myCar2.model = "Cybertruck";
        myCar2.year = 2024;
        myCar2.fuelEfficiency = 34.0;  // MPGe for electric cars
```

```
//compute range of first car (assuming 15-gallon tank)
        range = myCar1.fuelEfficiency * 15;
System.out.println("A " + myCar1.year + " " + myCar1.make + " " + myCar1.model
+ " can travel approximately " + range + " miles on a full tank.")

// compute range of second car (assuming 75 kWh battery)
        range = myCar2.fuelEfficiency * (75 / 33.7);
        System.out.println("A " + myCar2.year + " " + myCar2.make + " " +
myCar2.model + " can travel approximately " + range + " miles on a full charge.");
```

```
A 2024 Tesla Roadster can travel approximately 7.5 miles on a full tank.
A 2024 Tesla Cybertruck can travel approximately 75.66765578635014 miles on a full charge.
```

Declaring Objects

```
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```



```
Car myCar;
```

```
myCar = new Car();
```

Declaring an object of type Car

Declare object

The first line declares myCar as a reference to an object of type Car.

At this point, myCar does not yet refer to an actual object.

The next line allocates an object and assigns a reference to it to myCar.

myCar simply holds the memory address of the actual Car object.

new operator

- The new operator dynamically allocates memory for an object.

```
class-var = new classname ( );
```

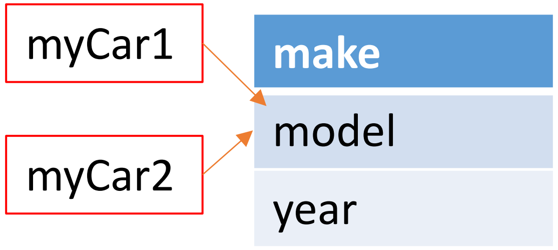
- The classname is the name of the class that is being instantiated.
- The class name followed by parentheses specifies the constructor for the class.
- A **constructor** defines what occurs when an object of a class is created.
- Most real-world classes explicitly define their own constructors within their class definition.
- However, if no explicit constructor is specified, then Java will automatically supply a default constructor.

Assigning Object Reference Variables

```
Car c1 = new Car();
Car c2 = c1;
```

- c1 and c2 will both refer to the same object.
- The assignment of c1 to c2 did not allocate any memory or copy any part of the original object.
- It simply makes c2 refer to the same object as does c1.
- Thus, any changes made to the object through b2 will affect the object to which c1 is referring, since they are the same object.
- Assign one object reference variable to another object reference variable, is not creating a copy of the object, only making a copy of the reference.

```
Car myCar1 = new Car();
Car myCar2=mycar1
```



```
Car c1 = new Car();
 Car c2 = c1;
// …
c1 = null;
```

Here, c1 has been set to null, but c2 still points to the original object.

Introducing Methods

- Classes usually consist of two things:
  - instance variables
  - methods.
- This is the general form of a method:

```
type name(parameter-list) {
// body of method
return value;
 }
```

Type

Type specifies the type of data returned by the method.

This can be any valid type, including class types

If the method does not return a value, its return type must be void.

Name

The name of the method is specified by name.

This can be any legal identifier .

Parameter-list

The parameter-list is a sequence of type and identifier pairs separated by commas.

Parameters are essentially variables that receive the value of the arguments passed to the method when it is called.

If the method has no parameters, then the parameter list will be empty.

Return

Methods that have a return type other than void return a value to the calling routine using the return statement:

Adding a Method to the Car Class

Use methods to access the instance variables defined by the class.

methods define the interface to most classes.

This allows the class implementor to hide the specific layout of internal data structures behind cleaner method abstractions.

In addition to defining methods that provide access to data, we can also define methods that are used internally by the class itself.

Instance variable that is not part of the class, it must be accessed through an object, by use of the dot operator.
Instance variable that is part of the same class, that variable can be accessed directly.
The same thing applies to methods.

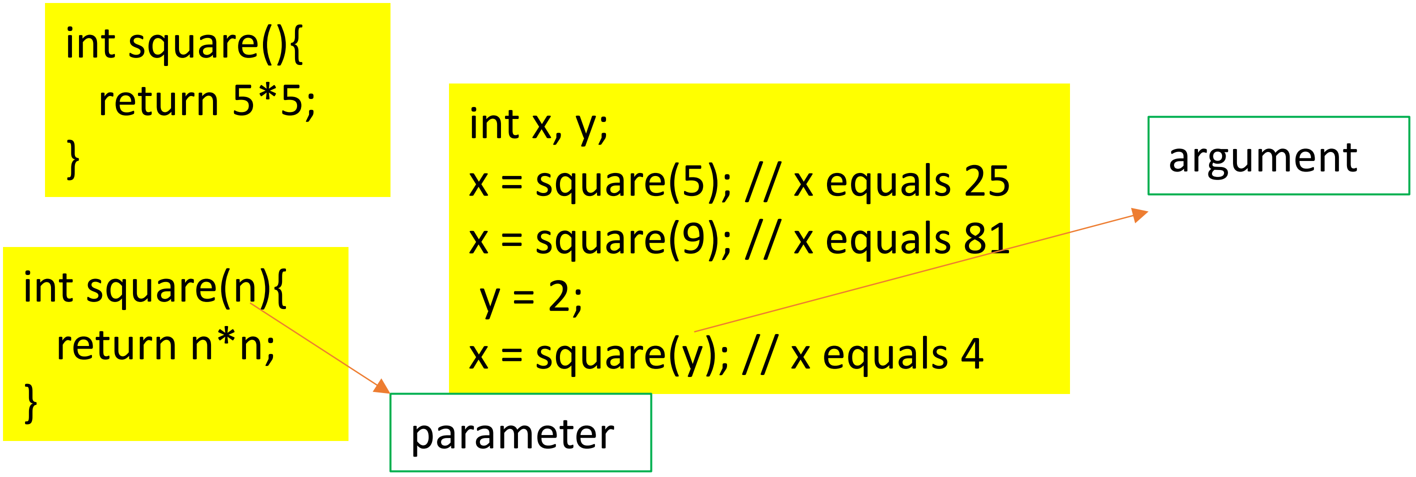
```
class Car {
   String make;
   String model;
   int year;
   double fuelEfficiency;  // miles per gallon
   // Method to calculate the range of the car
   public double calculateRange(double tankSize
      return fuelEfficiency * tankSize;
   }
} class CarDemo {
   public static void main(String args[]) {
      Car myCar = new Car();
      double tankSize = 15.0; // Size of the fuel tank in gallons
      myCar.make = "Tesla";
      myCar.model = "Roadster";
      myCar.year = 2024;
      myCar.fuelEfficiency = 0.5; // miles per gallon

      // Compute the range of the car using the new method
      double range = myCar.calculateRange(tankSize);
System.out.println("A " + myCar.year + " " + myCar.make + " " +
myCar.model + " can travel approximately " + range + " miles on a full
tank.");
```

No dot operator to access the class instance variable within class

dot operator to access the class instance variable outside the class

```
A 2024 Tesla Roadster can travel approximately 7.5 miles on a full tank.
```

Adding a Method That Takes Parameters



```
int square(){
   return 5*5;
}
```

```
int square(n){
   return n*n;
}
```

```
int x, y;
x = square(5); // x equals 25
x = square(9); // x equals 81
 y = 2;
x = square(y); // x equals 4
```

- A parameterized method can operate on a variety of data.
- A non parameterized method use is very limited

<u>parameter and argument:</u>

- A parameter is a variable defined by a method that receives a value when the method is called.
- For example, in square( ), i is a parameter.
- An argument is a value that is passed to a method when it is invoked.
- For example, square(100) passes 100 as an argument.

Constructors

- A constructor initializes an object immediately upon creation.
- Automatic initialization of object is performed through the use of a constructor.
- Constructor has the same name as the class in which it resides and is syntactically similar to a method.
- Once defined, the constructor is automatically called when the object is created, before the new operator completes.
- Constructor’s have no return type, not even void.
  - The implicit return type of a class’ constructor is the class type itself.

```
Car mybox1Car = new Car();
```

The parentheses are needed after the class name is to invoke the constructor for the class is being called.

Default vs defined constructor

- Java creates a default constructor for the class if a constructor is not explicitly defined for a class.
- The default constructor will initialize all non initialized instance variables to their default values.
  - zero, null, and false, for numeric types, reference types, and boolean.
- Once define your own constructor, the default constructor is no longer used.

```
    String make;
    String model;
    int year;
    double fuelEfficiency;
    // No-argument constructor
    public Car() {
        // Default values
        this.make = "Unknown";
        this.model = "Unknown";
        this.year = 0;
        this.fuelEfficiency = 0.0;
    }
    // Parameterized constructor
public Car(String make, String model, int year, double fuelEfficiency)
{
        this.make = make;
        this.model = model;
        this.year = year;
        this.fuelEfficiency = fuelEfficiency;
    }
    // Method to calculate the range of the car
    public double calculateRange(double tankSize) {
        return fuelEfficiency * tankSize;
    }}
```

```
Default Car:
Make: Unknown
Model: Unknown
Year: 0
Fuel Efficiency: 0.0 miles per
gallon

A 2024 Tesla Roadster can travel
approximately 7.5 miles on a full
tank.
```

```
class CarDemo {
    public static void main(String[] args) {
        // Using the no-argument constructor
        Car defaultCar = new Car();
        System.out.println("Default Car:");
        System.out.println("Make: " + defaultCar.make);
        System.out.println("Model: " + defaultCar.model);
        System.out.println("Year: " + defaultCar.year);
        System.out.println("Fuel Efficiency: " +
defaultCar.fuelEfficiency + " miles per gallon");

        // Using the parameterized constructor
        Car myCar = new Car("Tesla", "Roadster", 2024,
0.5);
        double tankSize = 15.0; // Size of the fuel tank in
gallons
         double range = myCar.calculateRange(tankSize);
        System.out.println("\nA " + myCar.year + " " +
myCar.make + " " + myCar.model +  " can travel
approximately " + range + " miles on a full tank.");
    }
}
```

The this Keyword

- **this keyword** allows a method to refer to the object that invoked it.
- **this** can be used inside any method to refer to the current object.
- **this** is always a reference to the object on which the method was invoked.
- use this anywhere as a reference to an object of the current class' type.

```
// no name space collision but this is used as redundant
 public Car(String a, String b, int c, double d) {
 this.make = a;
 this.model = b;
 this.year = c;
 this.fuelEfficiency = d; }
```

```
// no name space collision so no need of this operator
 public Car(String a, String b, int c, double d) {
 make = a;
 model = b;
 year = c;
 fuelEfficiency = d; }
```

this keyword:Instance Variable Hiding

- It is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes.
- There can be Overlap of local variables, including formal parameters to methods with the names of the class' instance variables.
- However, when a local variable has the same name as an instance variable, **the local variable hides the instance variable**.
- this directly refers to the object,
- use this to resolve any namespace collisions that might occur between instance variables and local variables.

```
// use this operator to avoid name space collision
 public Car(String make, String model, int year, double fuelEfficiency) {
this.make = make;
 this.model = model;
.year = year;
this.fuelEfficiency = fuelEfficiency; }
```

```
public class Stack {
    private char[] vowel;
    private int top;
    private static final int INITIAL_CAPACITY = 10; // Initial
public Stack() {
        vowel = new char[INITIAL_CAPACITY];
        top = -1;
    }
    public void push(char c) {
        if (c == 'a' || c == 'e' || c == 'i' || c == 'o' || c == 'u’ ||
            c == 'A' || c == 'E' || c == 'I' || c == 'O' || c == 'U') {
            if (top == vowel.length - 1) {
                System.out.println("Stack is full");
            }
            vowel[++top] = c;
        } else {
            System.out.println(c + " is not a vowel.");
        }       }
   public Character pop() {
        if (top == -1) {
            System.out.println("Stack is empty.");
            return null;
        }
        return vowel[top--];        }
```

```
// Method to peek the top vowel of the stack
    public Character peek() {
        if (top == -1) {
            System.out.println("Stack is empty.");
            return null;
        }
        return vowel[top];
    }

public static void main(String[] args) {
        Stack vowelStack = new Stack();
        vowelStack.push('a');
        vowelStack.push('b');   // Not a vowel
        vowelStack.push('e');
        vowelStack.push('i');
        System.out.println("Top of the stack: " + vowelStack.peek());
        System.out.println("Popped: " + vowelStack.pop());
        System.out.println("Popped: " + vowelStack.pop());
        System.out.println("Popped: " + vowelStack.pop());
        System.out.println("Popped: " + vowelStack.pop());
    }
  }
```

```
b is not a vowel.
Top of the stack: i
Popped: i
Popped: e
Popped: a
Stack is empty.
Popped: null
```

Garbage collections

- In java, since objects are dynamically allocated by using the new operator, such objects are destroyed and their memory released for later reallocation automatically.
- In some languages, such as traditional C++, dynamically allocated objects must be manually released by use of a delete operator.
- It works like this: when no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed.
- There is no need to explicitly destroy objects.
- Garbage collection only occurs sporadically (if at all) during the execution of your program.