Julio Vasquez
1/27/2024
Computer Science 2

Technical Documentation
Characters and Strings(Manipulation and Operations)

Overview:

- **Characters**: Individual symbols like letters (a, B), digits (5), or special characters (@, #) are the building blocks of strings.
- **Strings**: A sequence of characters treated as a single data type, often enclosed in quotes

Examples:

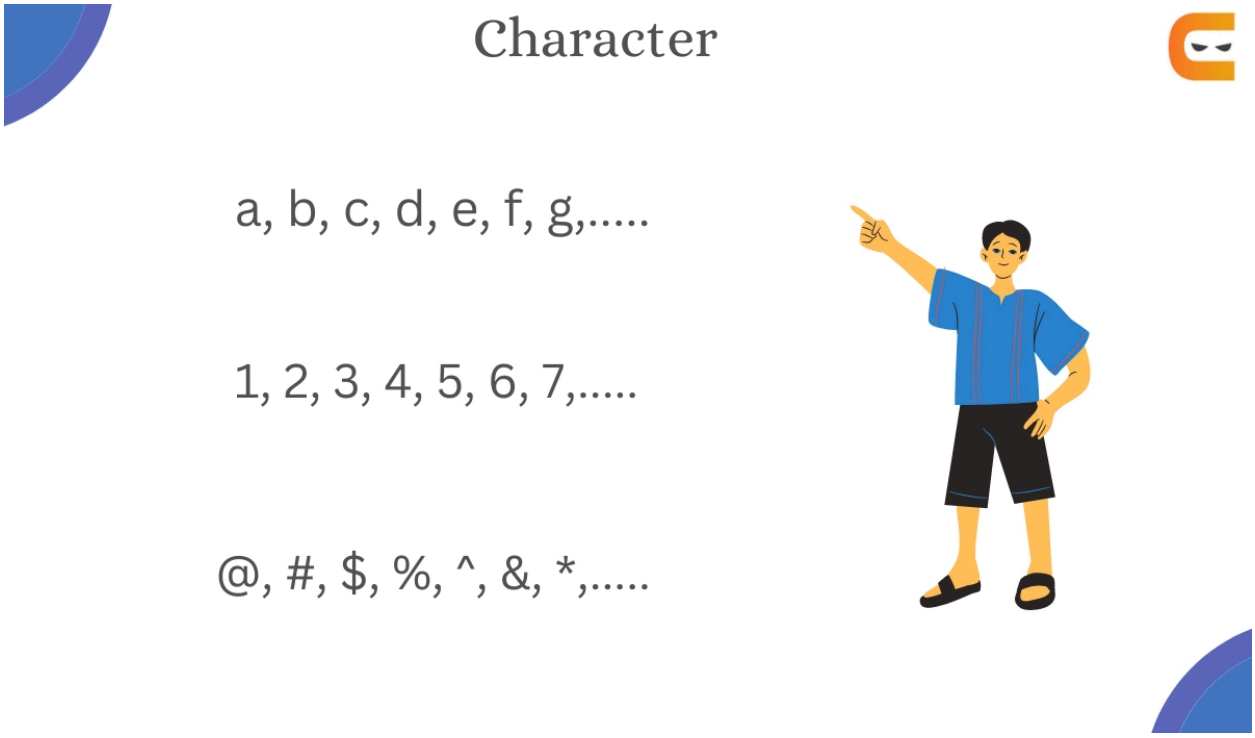
1: Characters in Programming:
Java:

```
char ch = 'A';
System.out.println(Character.isLetter(ch)); // true
System.out.println(Character.isDigit(ch));  // false
System.out.println(Character.toLowerCase(ch)); // 'a'
```

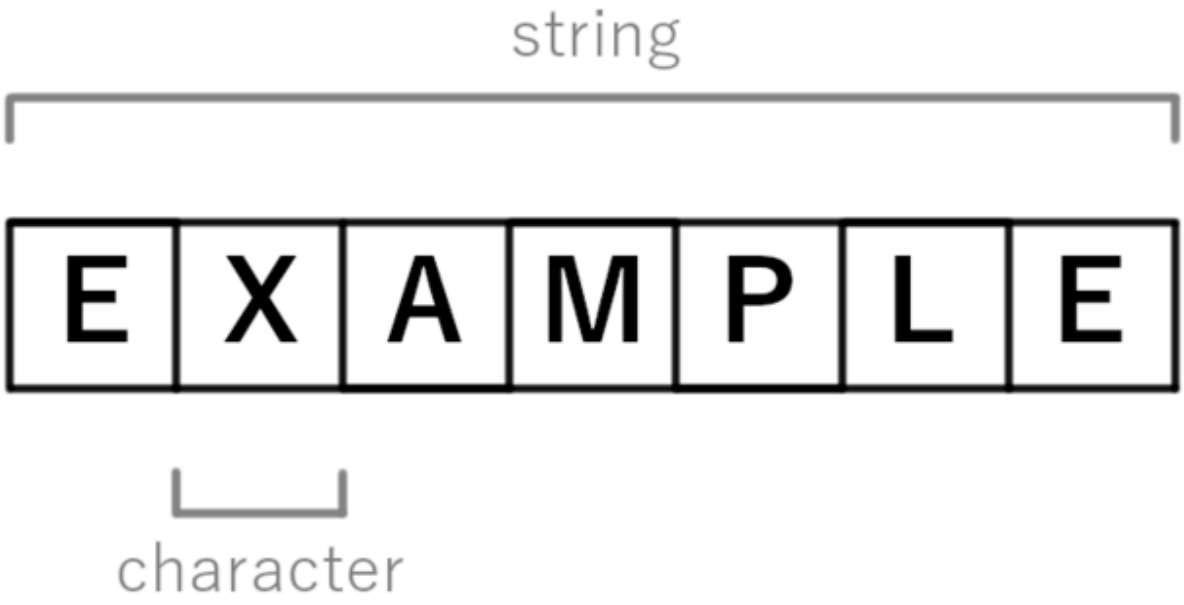
2: Strings in Programming:
Java:

```
String greeting = "Hello, " + "World!";
System.out.println(greeting);  // Output: Hello, World!
```

Visual:



String:



Resources:

- https://docs.oracle.com/javase/tutorial/java/data/stringsummary.html
- https://stackoverflow.com/questions/10430043/difference-between-char-and-string-in-java

- https://docs.oracle.com/javase/tutorial/java/data/characters.html
-

Summary: Characters and strings are core to programming, enabling the handling of textual data. Strings offer powerful methods for manipulation, such as concatenation, slicing, searching, and replacing. Understanding and mastering these operations is essential for data processing, user input handling, and more advanced applications like regular expressions and string formatting. With the resources and examples provided, you can further explore and practice string manipulation in different programming languages.

Classes vs Objects

difference between a class and an object

Overview: In object-oriented programming (OOP), a **class** and an **object** are fundamental concepts. A **class** is a blueprint or template that defines the attributes (fields) and behaviors (methods) of an object. An **object** is an instance of a class, containing actual values that conform to the structure defined by the class.

Example:

1: Classes in Programming

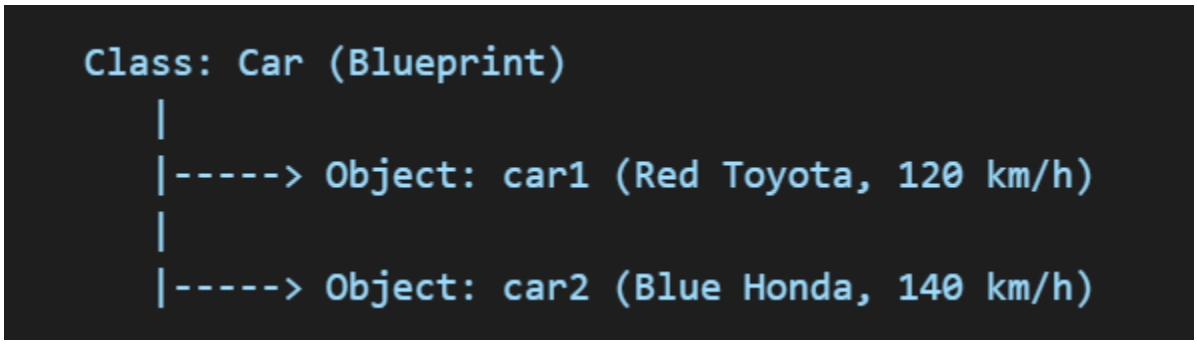
```
// Defining a Class

class Car {

//Instance Variables

   String color;

   String model;

   int speed;


   // Constructor

   Car(String color, String model, int speed) {

      this.color = color;
```

```
        this.model = model;
        this.speed = speed;
    }

    // Method to display car details
    void displayCar() {
        System.out.println("Car Model: " + model + ", Color: " + color + ", Speed: " + speed + "
km/h");
    }
}
```

2: Objects in Programming:

```
// Creating Objects
public class Main {
    public static void main(String[] args) {
        Car car1 = new Car("Red", "Toyota", 120); //”new” used to make a new object/instance
of car
        Car car2 = new Car("Blue", "Honda", 140);

        car1.displayCar();
        car2.displayCar();
    }
}
```

Visual:



Summary

- **class is a template for creating objects, defining attributes and behaviors.**
- **An object is an instance of a class, holding real values.**
- **Classes do not occupy memory until objects are created.**
- **Objects execute methods and interact with other objects based on class definitions.**

Understand state and behavior are represented (what other terms are used)

Overview: In object-oriented programming (OOP), **state** and **behavior** are two fundamental aspects that define objects. **State** represents the data (attributes or properties) an object holds, while **behavior** represents the actions (methods or functions) the object can perform. These concepts are crucial for designing effective OOP systems.

Examples:

State in Programming:

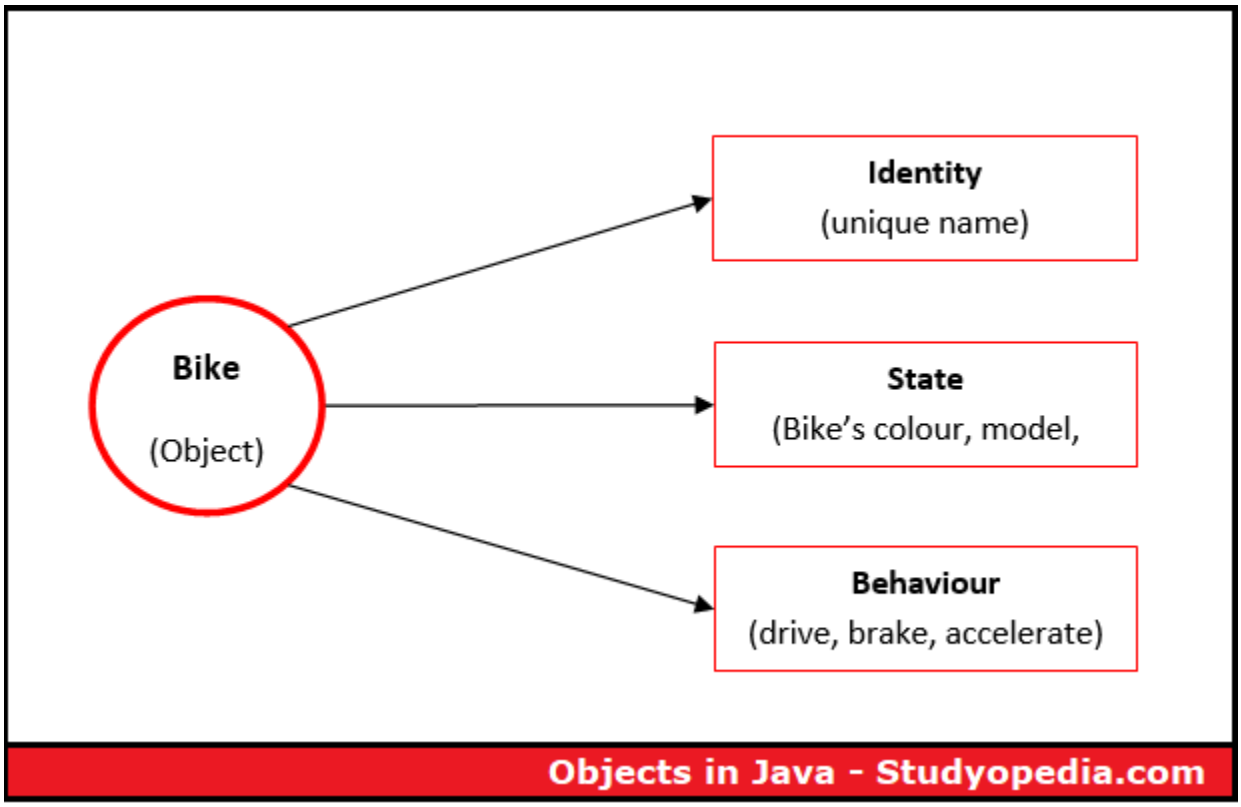
```
// Defining a Class with State and Behavior

class Car {

   String color; // State

   String model; // State

   int speed;    // State
```

Behavior in Programming:

```
 // Behavior: Method to display car details
```

```
    void displayCar() {

        System.out.println("Car Model: " + model + ", Color: " + color + ", Speed: " + speed + "
km/h");

    }

}
```

Visual:



Summary

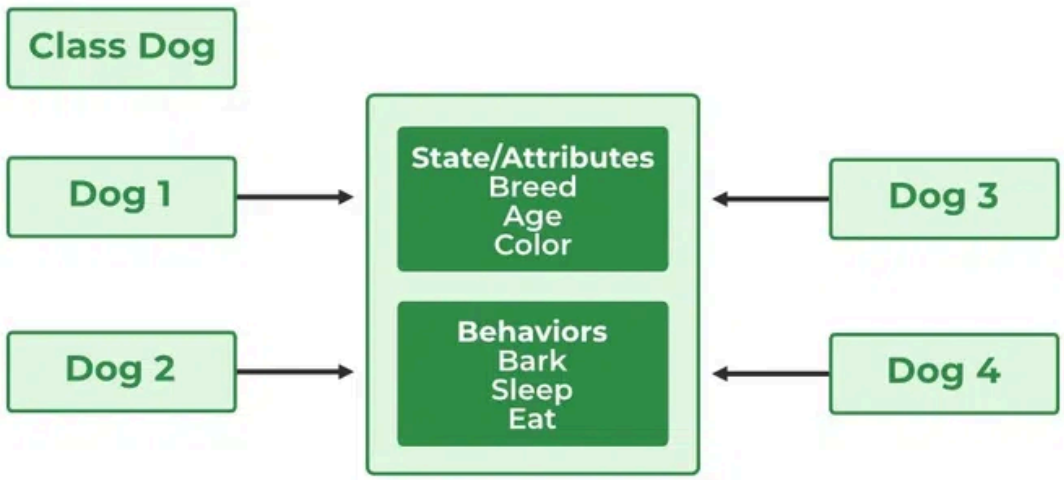
- **State** refers to an object's attributes (properties, data).
- **Behavior** refers to an object's methods (actions, functions).
- Attributes and methods together define how an object behaves in an OOP system.
- Understanding state and behavior helps in designing robust and scalable applications.

Know how to create an object from a class (i.e., know what instantiation means)

Overview: In order to make an object from a class you have to use the “new” key word which is then followed by a class constructor if needed.

Examples:(Code Snippet)
```
// Create an object of the Car class
Car myCar = new Car("Toyota", "Corolla");
```

Visual:



Sources:

Constructors:

Overview:

A constructor in Java is a method that is used to initialize objects. It is automatically called when an instance of a class is created. The main purpose of a constructor is to set initial values for object attributes.

What a Constructor Is and What It Does:

- Is a method that has the same name as the class.
- Does not have a return type (not even void).
- Can be used to initialize object fields.
- Is invoked automatically when an object is created.

Differences Between a Constructor and a Regular Method

Feature	Constructor	Regular Method
Name	Same as Class Name	Any method name
Return Type	No return type	Can return a value of be void
Invocation	Called automatically when an object is instantiated	Expliciltly called using an object
Purpose	Initalizes an object	Peforms Actions on an Object

Default Constructor and Overloaded Constructor

Default Constructor:

If no constructor is used then Java will then provide a **default constructor** that initializes instance variables to default values (e.g., 0 for numbers, null for objects, false for booleans).

Coding Example:

Class Student {

String Name;

Int Birthday_Year;

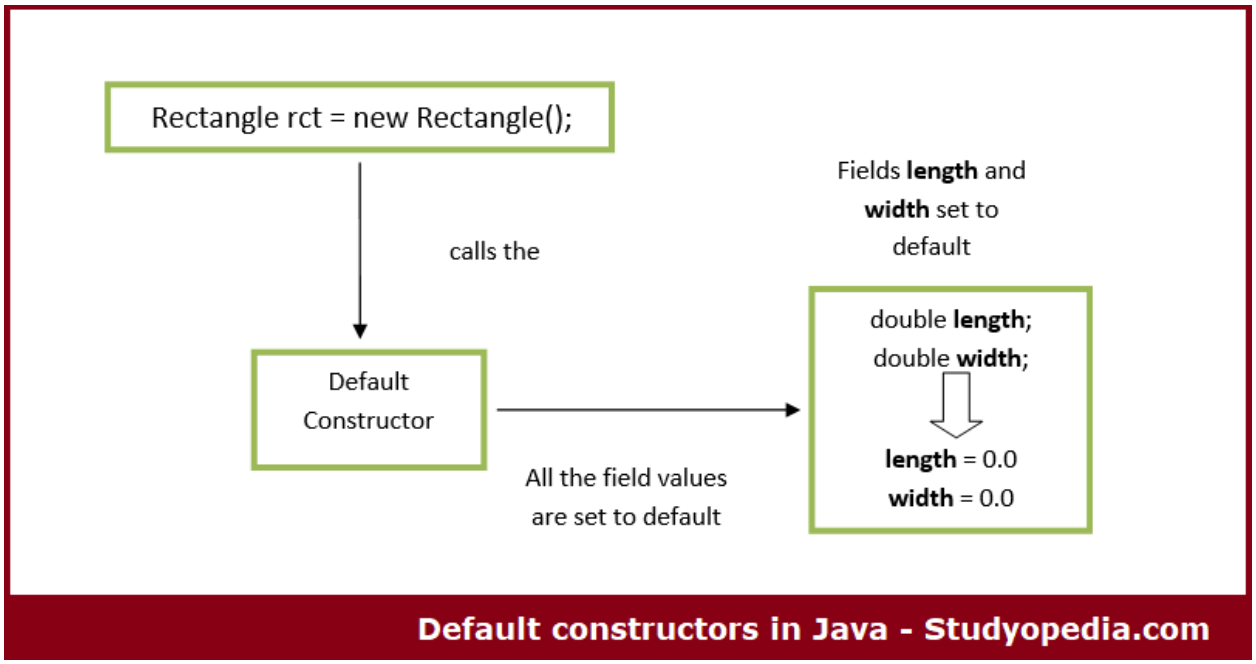
//Default Constructor Outcomes

Public Student() {

this.Name = “unknown” //null

this.Student_Year = 0

Visual:



Overloaded Constructor

Constructors can have multiple constructors with different parameters(lists).

Coding Example:

 Class Dog {

String Breed;

Int Age; //in years;

//Constructor with No Paramters(default)

Public Dog() {

Breed = “Unknown”;

Age = 0’’; //Default Age

//Constructor With One Parameter(Breed)

Public Dog(String Breed) {

This.Breed = Breed;

This.Age = 0

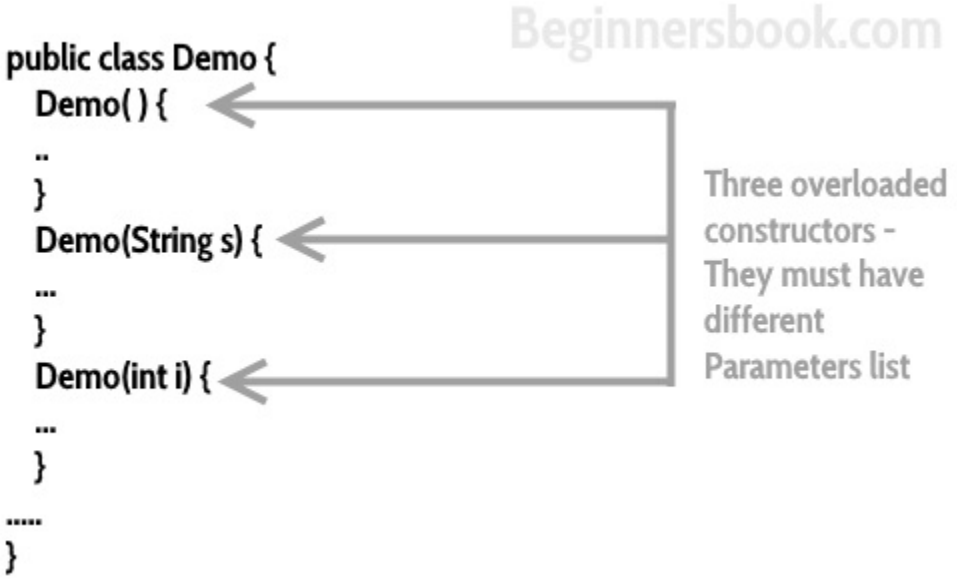
//Constructor with Two Parameters(Breed, Age)

Public Dog(String Breed, Int Age) {

this.breed = Breed

this.Age = Age

Visual:



Constructor Invocation Using new and Memory Allocation

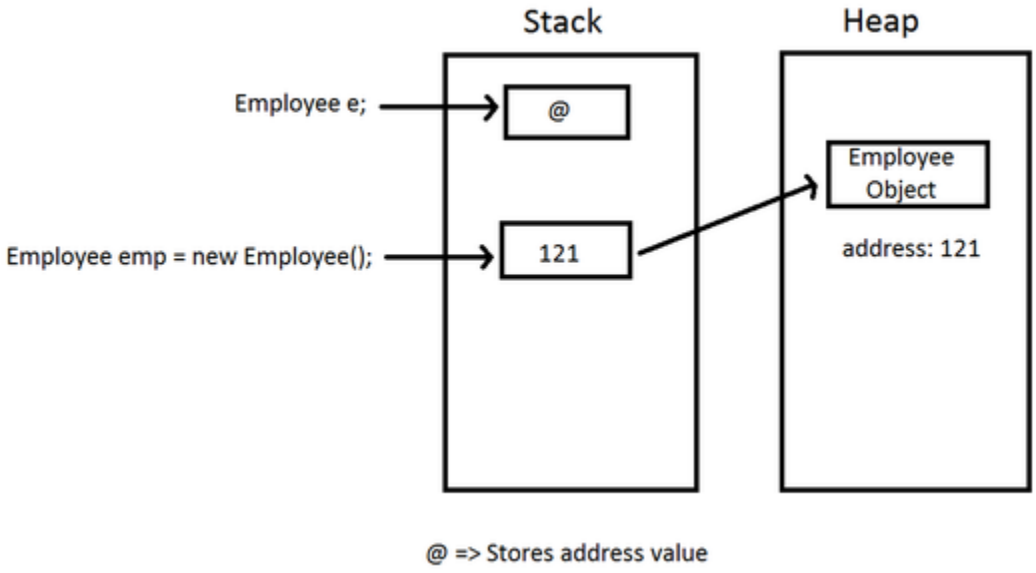
When an object is instantiated using new, memory is allocated on the heap, and a reference variable is stored on the stack.

Coding Example:

Dog Dog1 = new Dog("Golden Retriever", 3); //Dog Breed and Age in Years

- New Dog("Golden Retriever", 3) Creates a new dog object on the **Heap**
- Dog1 is a reference variable that becomes stored on the **Stack** which will also store the memory address of that specific object that was made

Visual:



What Are Reference Variables?

A **reference variable** holds the memory address of an object rather than the object itself. When a method is called/used on a reference variable it will refer to the location where the memory is stored. It is used to access objects

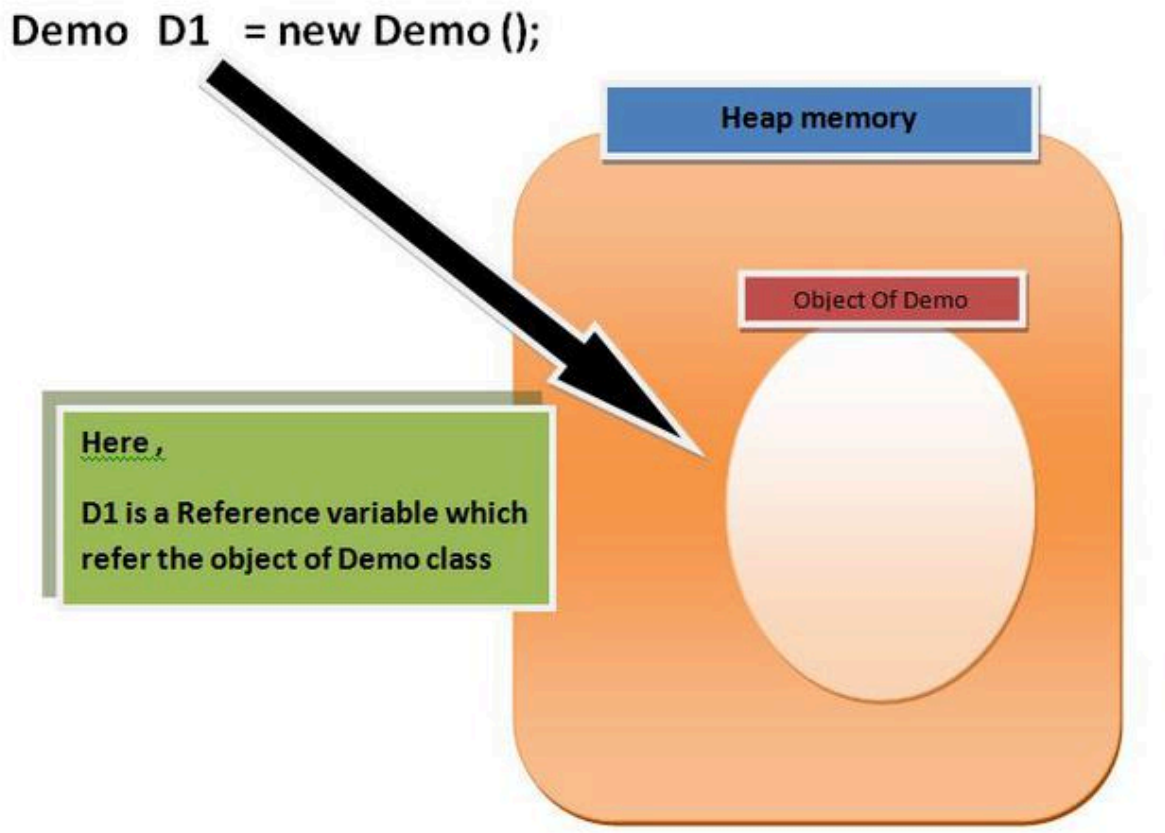
Coding Example:

Dog Dog1 = new Dog("Golden Retriever", 3); //Dog1 is the reference variable

Coding Example of Reference Variable Being Used

Dog Dog2 = Dog1 //Dog2 will now reference Dog1 as the same object

Visual:

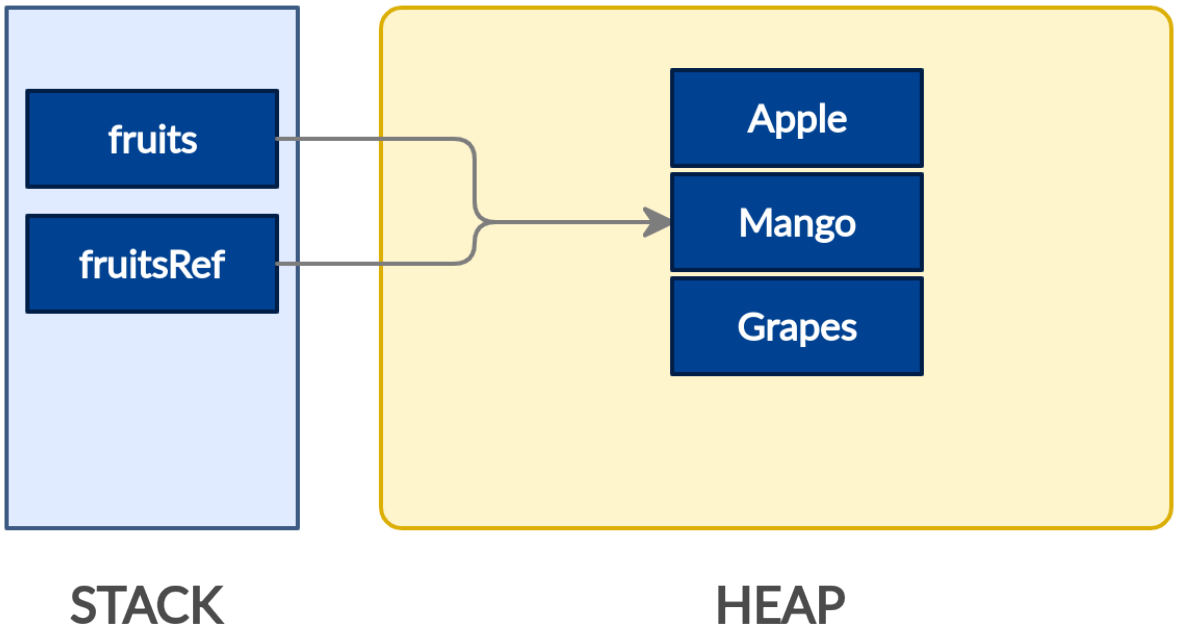


How Are Objects Passed?

Objects in Java are passed by reference. When an object is passed as a parameter, the reference (memory address) is copied, meaning any modifications affect the original object.

Coding Example:

Visual:



Resources:

- https://docs.oracle.com/javase/tutorial/java/javaOO/constructors.html
- https://www.geeksforgeeks.org/constructors-in-java/
- **https://youtu.be/5DdacOkrTgo?si=ErNCWCyCdZ09wO_f**

Summary

- **A constructor initializes an object when it is created.**
- **Different from methods, constructors have no return type and share the class name.**
- **Java provides a default constructor if none is defined.**
- **Overloaded constructors allow different ways to initialize an object.**
- **Using new invokes a constructor and allocates memory.**
- **Reference variables store memory addresses of objects.**
- **Objects are passed by reference, meaning changes persist outside methods.**

Understanding constructors is crucial for writing efficient, object-oriented Java programs!

Instance and Static Variables and Methods

Overview: In object-oriented programming (OOP), variables and methods are key concepts used to define the behavior and attributes of objects and classes. These elements can be classified into two categories: instance and static. Understanding the distinction between them is crucial for effective programming.

Instance Variables and Methods

- **Instance Variable: Variables that are connected or associated with a instance of a class or object.**
- **Instance Method:** local
- Instance methods require an object of the class in order to be called and referenced in a line of code

Static Variables and Methods

- **Static Variable:** These are variables that can be shared and used across multiple/all instances of a class. There is only one copy of this variable that is saved.
- **Static Method:** Can be called without the class
- Static methods are associated with the class itself, not with any particular instance of the class.

Instance Variables and Methods

When to Use:

- **Instance Variables**: Use instance Variables whenever you want each of your objects to have their own data that isn't being used by another object
- Ex: If you have a Dog class each object of that Dog Class might have their own Breed, Age, Height attributes with different values
- **Instance Methods:** Use instance Methods when you need that method to interact with some data like an instance variable

Coding Example:

```
public Class Dog {

//Instance Variables
Private String Breed;
```

```
Private Int Age; //In Years

//Constructor
Public Dog (String Breed, Int Age) {
this.Breed = Breed;
this.Age = Age

//Instance Method
public void DisplayInfo() {
        System.out.println("Dog Breed: " + Breed + "Dog Age: " + Age);
```

Visual:

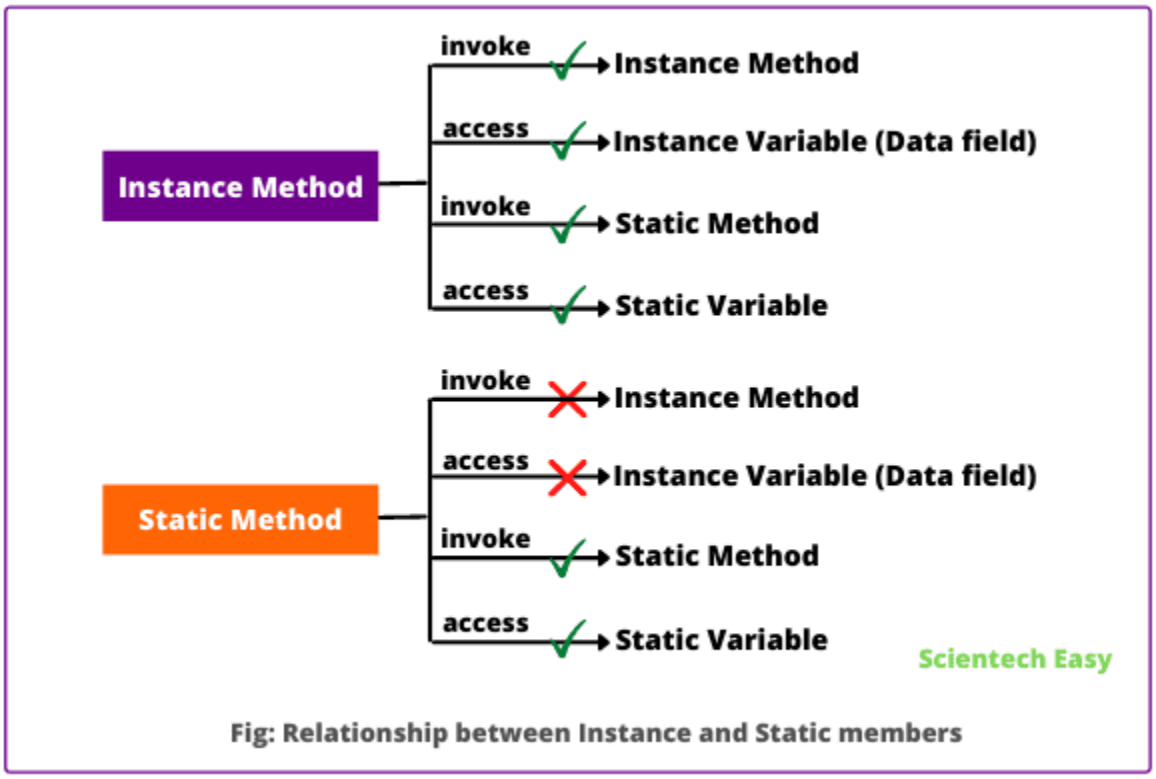


Fig: Relationship between Instance and Static members

Static Variables and Methods

When to Use

- Static Variables: Use Static Variables when you want to store/save a value that should be used across all objects or instances of a class.
- Static Methods: Use Static method when the method action does not need to interact or use an instance variable data and whenever it can be called without creating an object.

Coding Example:

```
//Static Variable
static double pi = 3.14159;

//Static Method to get year
static int square(int number) {
        return number * number;
```

Visual:

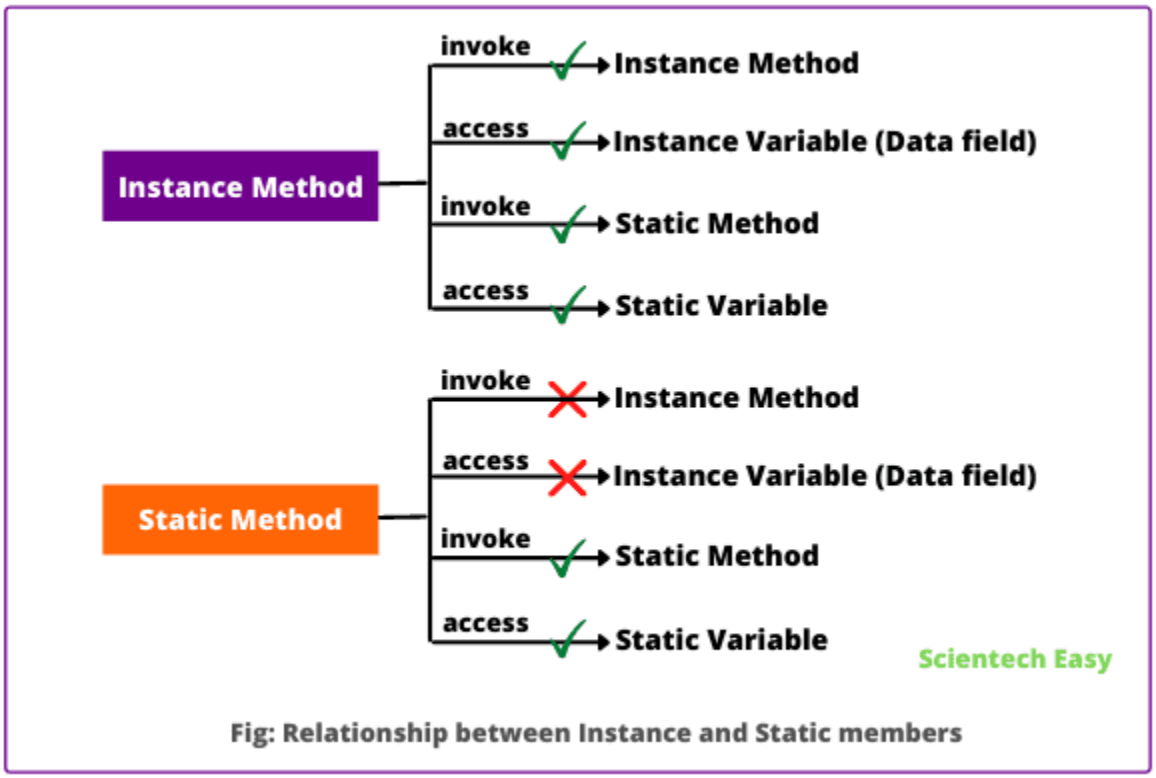


Fig: Relationship between Instance and Static members

Resources:

- https://docs.oracle.com/javase/tutorial/java/javaOO/variables.html
- https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html
- https://www.youtube.com/watch?v=opF7S3_p-2s
- https://www.youtube.com/watch?v=2g9eah4Tt6Q
-

Summary

- **Instance Variables and Methods**: Use when data and behavior belong to individual objects. Each object gets its own copy of instance variables.

- **Static Variables and Methods**: Use when data and behavior should be shared across all instances of a class. Static members are accessed through the class itself, not an instance.

Understanding when to use instance vs. static members is essential for efficient and effective object-oriented design.

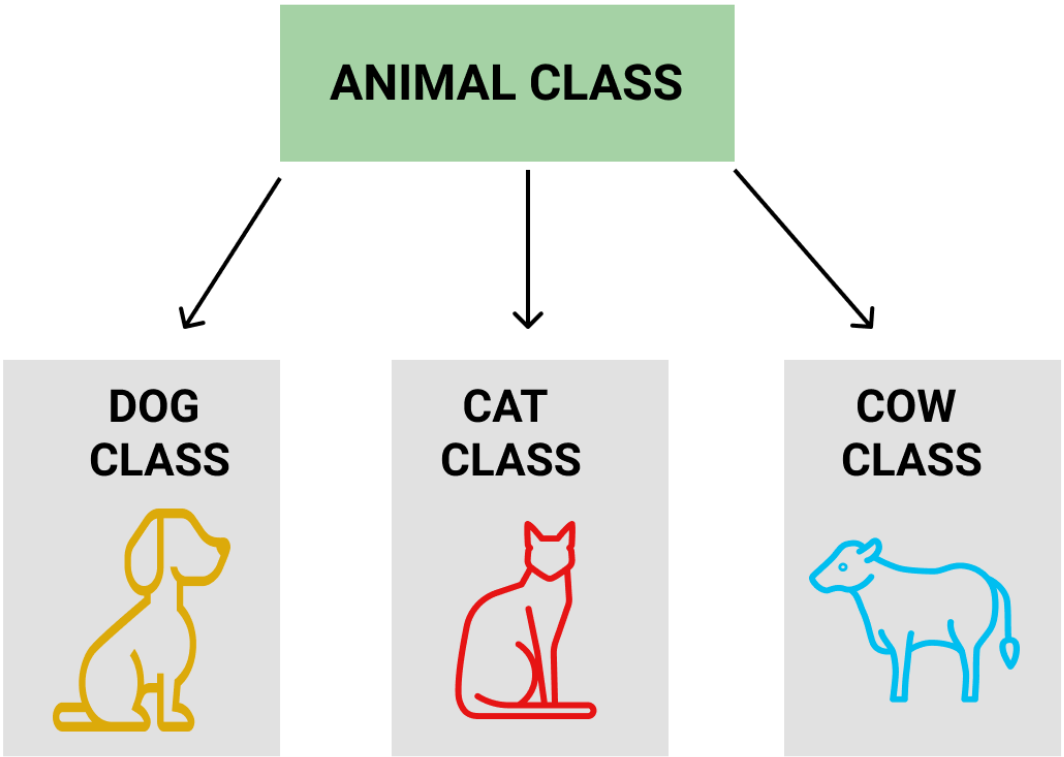
Inheritance

Overview: A system that allows one class(subclass or child class to inherit methods/behaviors and properties from another classs (superclass or parent class).

Coding Example:

```
class Car extends Vehicle {

   int wheels = 4;
```

Visual:



Sources: https://docs.oracle.com/javase/tutorial/java/IandI/subclasses.html

Summary: Inheritance is ultimately very useful as it allows a user to reuse and extend functionality of multiple classes and objects.

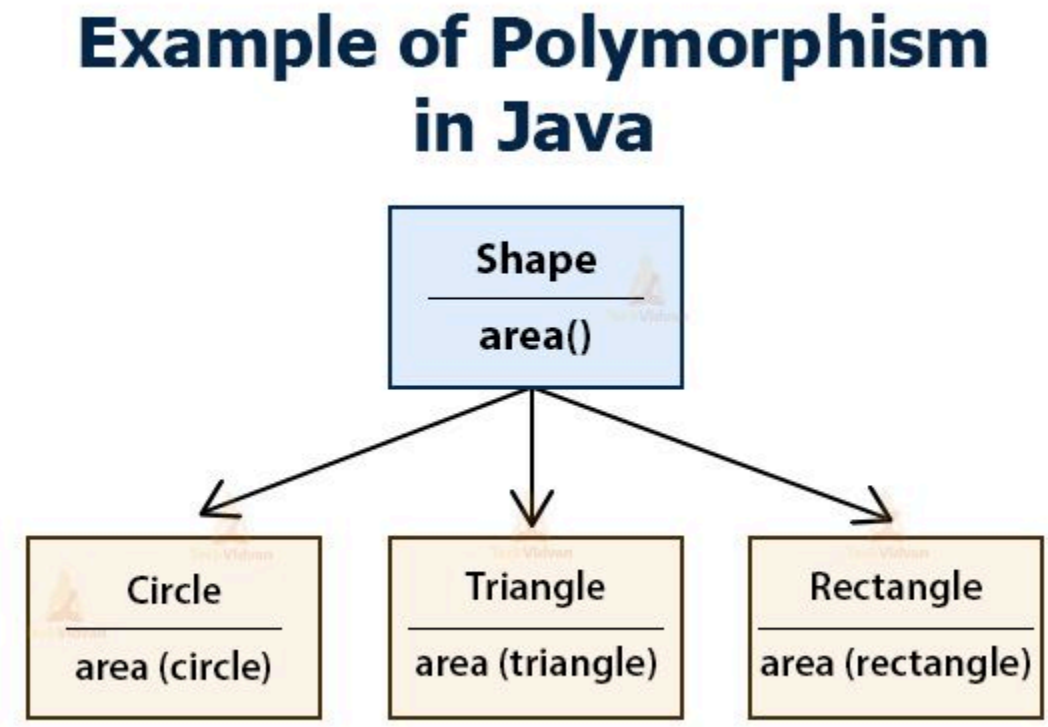
Polymorphism

Overview: Polymorphism allows objects of different classes to be objects of a single superclass. This enables and allows a lot of flexibility in code of programs. Polymorphism is achieved through method overloading (compile-time polymorphism) and method overriding (runtime polymorphism).

Coding Example:(from my own program)

```
// Subclasses representing specific animals
class Bear extends Animal {
  public Bear(String name, String food, int weight, int sleep, String location) {
     super(name, food, weight, sleep, location);
  }
  @Override
  public void swim() { System.out.println(getName() + " is swimming"); }
}
class Elephant extends Animal {
  public Elephant(String name, String food, int weight, int sleep, String location) {
     super(name, food, weight, sleep, location);
  }
}
```

Visual:



Sources:

- https://docs.oracle.com/javase/tutorial/java/IandI/polymorphism.html
- https://docs.oracle.com/javase/tutorial/
- https://www.geeksforgeeks.org/polymorphism-in-java/

Summary: Polymorphism in Java allows the same interface to be used for different data types, improving flexibility and code reusability.

Dynamic Binding

Overview: Dynamic binding is a mechanism in Java where method calls are resolved at runtime rather than at compile time

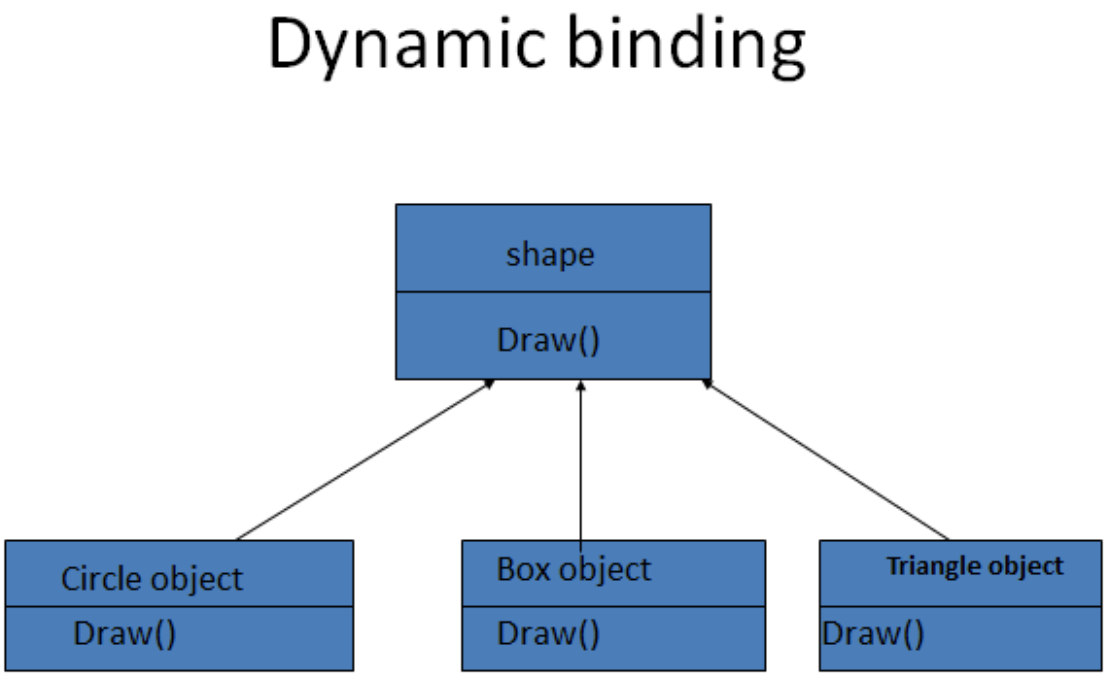
Code Example:

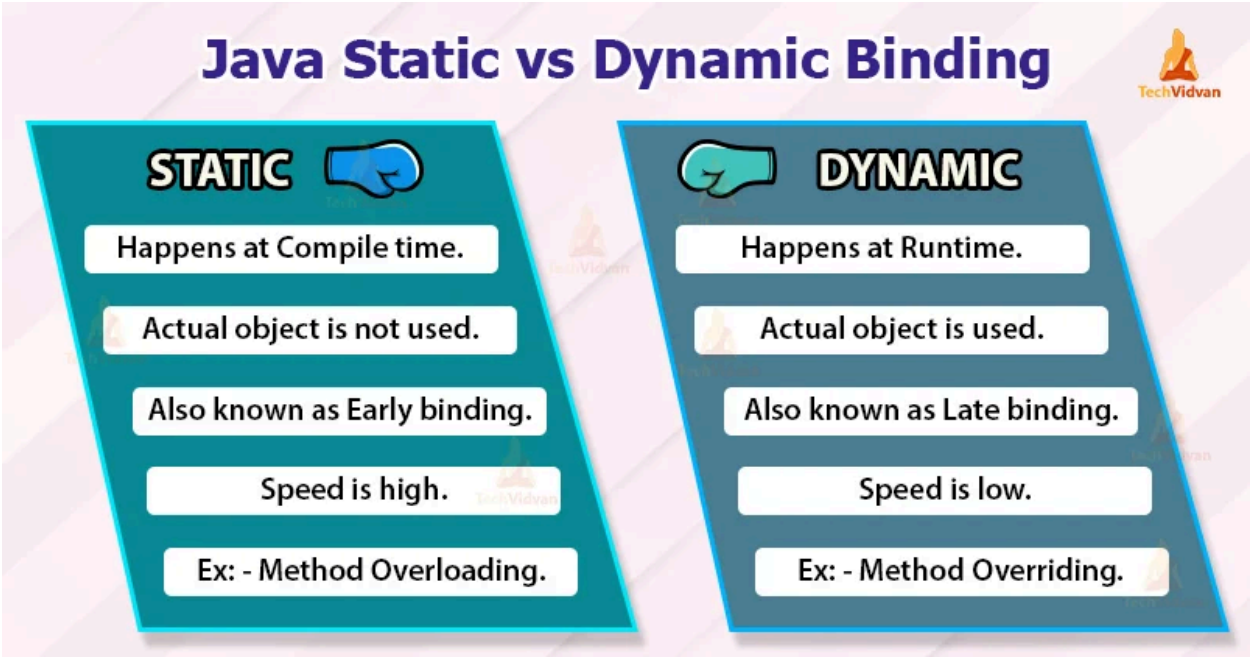
```
public static void main(String[] args) {
        Car MyCar = new Make(); // Dynamic binding
MyCar.moveCar() {  //calls the method during run time
```

Or

```
public static void main(String[] args) {
        Animal Animal1 = new Dog();
```

Visual Example:





Sources:

- https://docs.oracle.com/javase/tutorial/java/landl/polymorphism.html

- **https://docs.oracle.com/javase/tutorial/**
- **https://www.geeksforgeeks.org/static-vs-dynamic-binding-in-java/**

Summary: Dynamic binding in Java allows method calls to be resolved at runtime, making code more flexible and maintainable. It is achieved through method overriding and dynamic method dispatch, which allows a superclass reference to invoke subclass methods.