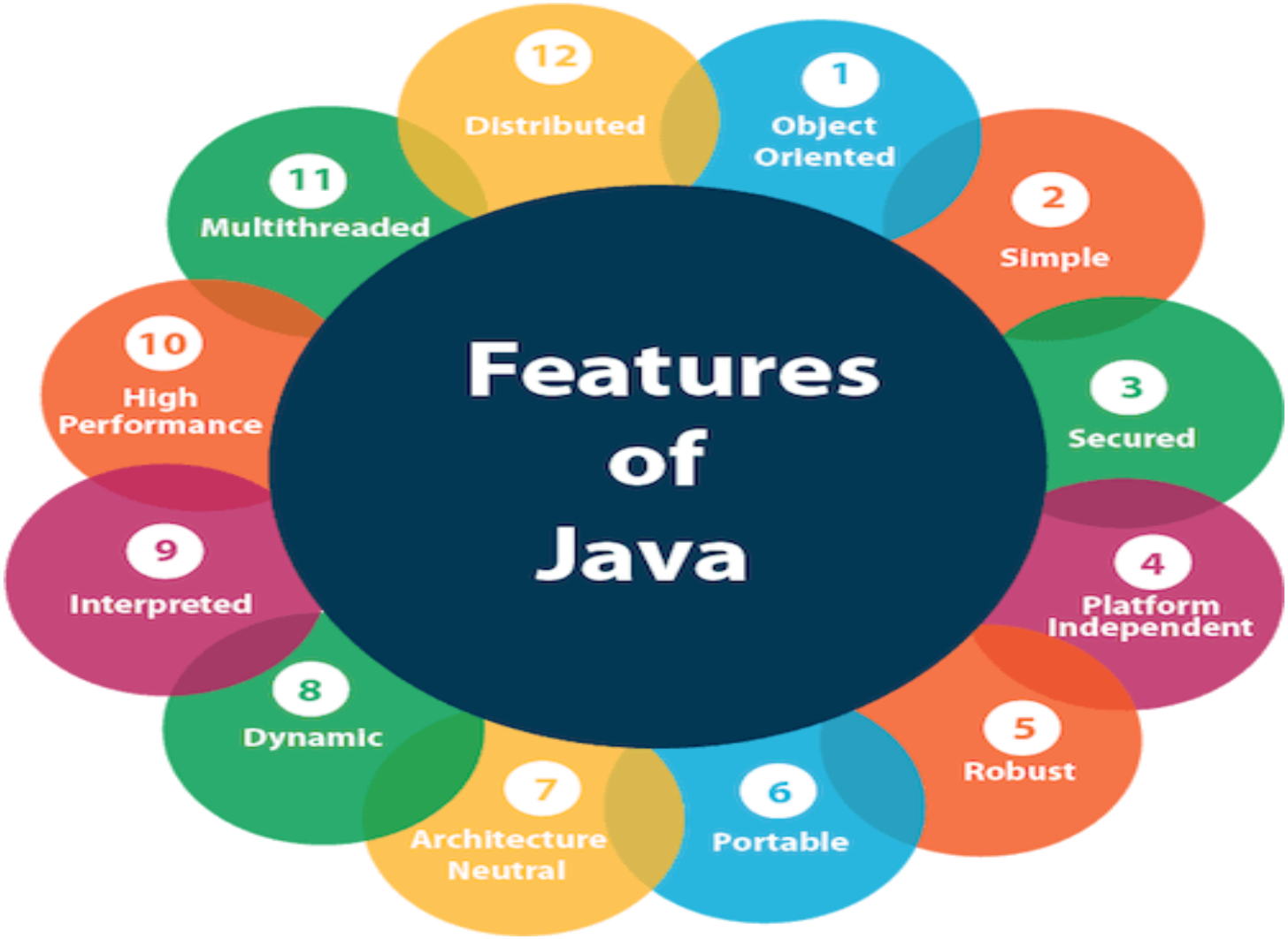
Object Oriented Programming with Java

Instructor : *Mr. Enock Wambi*

Introduction

- Java is an Object Oriented Programming (OOP) language.
- Object oriented programming paradigm relies on the concept of classes and objects
- It is also a platform- other programs can run on java. Since Java has a runtime environment (JRE) and API, it is called a platform,

Features of Java



Why Java

- Portability- Java code can execute on all major platforms
- Robustness- Uses strong memory management
- Simplicity- Simple, Clean and Easy to understand
- Secure programming –Runs in VM and byte code verifier checks the code fragments for illegal code that can violate access right to object.

OOP and its Concepts

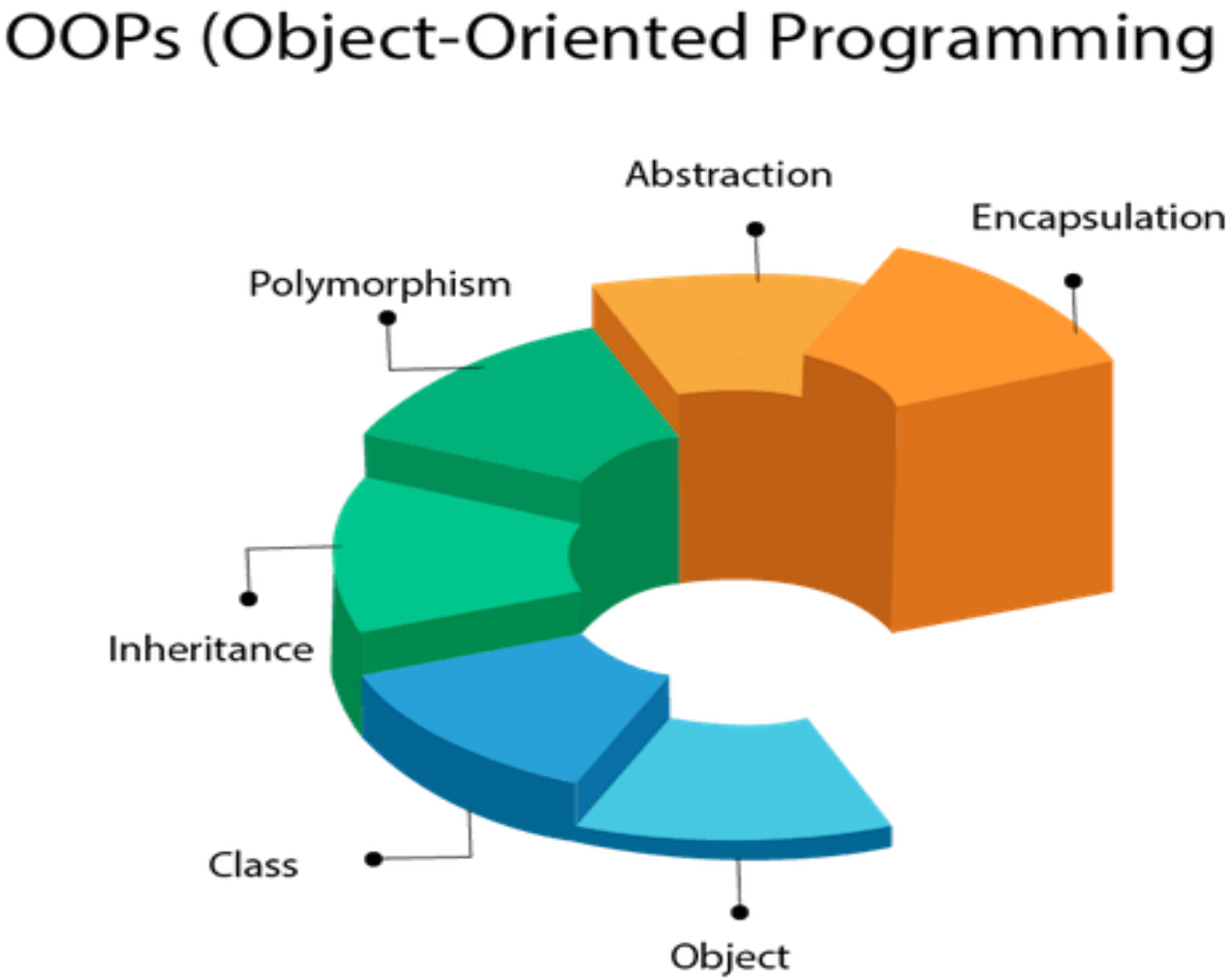
- **Object-Oriented Programming** is a methodology or paradigm to design a program using classes and objects.
- The basic idea of OOP is to divide a complex program into a bunch of **objects** talking to each other
- The basic entities in object-oriented programming are **classes and objects** **whereas the basic entities in procedural programming are methods**

OOP and its Concepts

It simplifies software development and maintenance by providing some concepts:

- Object and Class
- Abstraction
- Encapsulation
- Inheritance
- Polymorphism

OOP and its Concepts



Object

- Any entity that has state and behavior is known as an object.

  - An Object can be defined as an instance of a class. An object contains an address and takes up some space in memory.
  - Objects can communicate without knowing the details of each other's data or code .
  - The only necessary thing is the type of message accepted and the type of response returned by the objects.

Object (Example)

Objects may contain data in the form of *fields* (variables) and methods to operate on that data

For a while, think about the real-world objects around you. ***"What are the characteristics of these objects?***

Take the example of a *light bulb* as an object

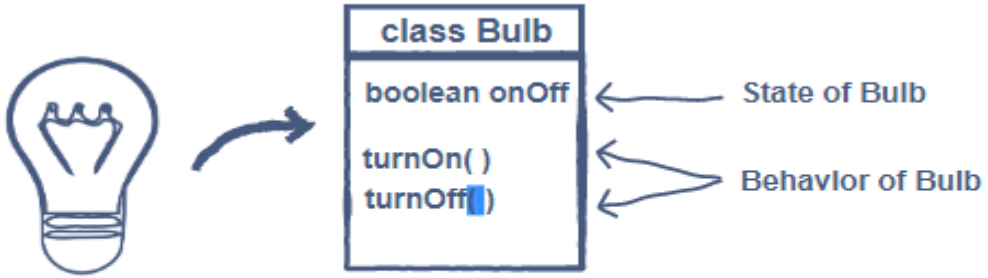
- It has a **state** i.e. either it is *on* or *off*.
- It also has a **behavior** i.e. when you turn it on it lights up and when turned off, it stops spreading light.

But the question is "*where do these objects come from?"*

Well, the answer to the above question is **classes**.

Class

A **Class** can be thought of as a *blueprint* for creating objects.



The state of the objects is generally modeled using *variables* in a class and the behavior is modeled by implementing the *methods*.

Student Class (Example)

We can create several objects of the type Student using a single class.

The Student class can have the data members (variabes) which represent the states of an object e.g. *student_name, student_ID, student_Age etc*.

We can have a method for initializing an object(storing value into the object) and also method for displaying object value.

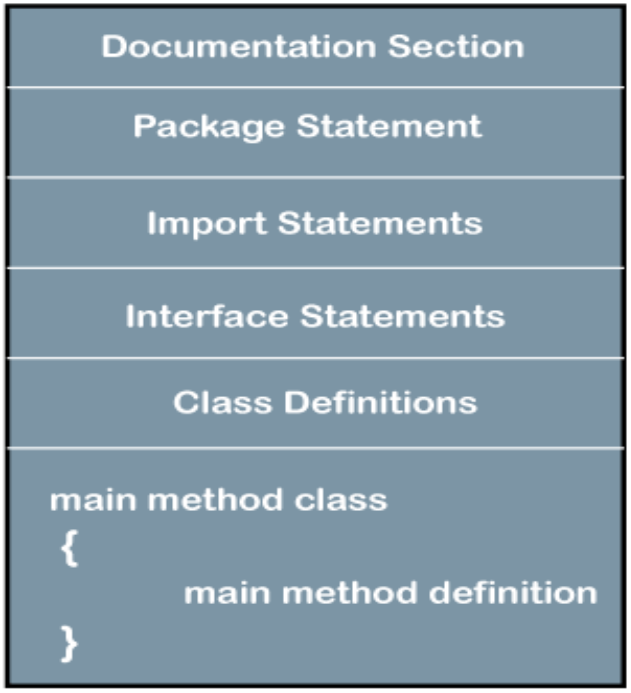
Example of an object in java that can be created from Student class

Student std1=new Student();

N.B std1 is an object of Student class and its value would be stored in std1.student_ID, std1.student_name and std1.student_Age

Structure of a Java Program

A basic java program contains of the following elements



Structure of Java Program

Structure of Java Program

1. Documentation

The documentation section is an important section but optional for a Java program. It includes **basic information** about a Java program. The information includes the **author's name, date of creation, version, program name, company name,** and **description** of the program

- It improves the readability of the program
- Whatever we write in the documentation section, the Java compiler ignores the statements during the execution of the program
- To write the statements in the documentation section, we use **comments**. The comments may be **single-line, multi-line,** and **documentation** comments.

The document section may include ;

```
// First Java program
```

single-line comment

```
/* This program computes the
Area of the rectangle*/
```

Multi-line comment

Structure of a Java Program

2. Package

The package declaration is optional. It is placed just after the documentation section. In this section, we declare the **package name** in which the class is placed. Note that there can be **only one package** statement in a Java program

```
package testrectangle;   // An example of package statement
```

3. Import Statements

The package contains the many predefined classes and interfaces. We use the **import** keyword to import the class. It is written before the class declaration and after the package statement. We use the import statement in two ways, either import a specific class or import all classes of a particular package.

```
import java.util.Scanner;      //it imports the Scanner class only
import java.util.*;      //it imports all the class of the java.util package
```

Structure of a Java Program

4.0 Interface Section

It is an optional section. We can create an **interface** in this section if required. We use the **interface** keyword to create an interface. An interface is a slightly different from the class. It contains only **constants** and **method** declarations.

```
interface car
{
void start();
void stop();
}
```

5.0 Class Definition

In this section, we define the class. It is **vital** part of a Java program. Without the class, we cannot create any Java program. A Java program may contain more than one class definition.

Structure of a Java Program

A class contains information about user-defined methods, variables, and constants. Every Java program has at least one class that contains the main() method. For example:

```
class Student //class definition
{
int ID;                    // An example of a variable of type integer
String name ;              // An example of variable of type string

}
```

5.1 Main Method

it is an entry point of the class. It must be inside the class. Within the main method, we create objects and call the methods. We use the following statement to define the main() method:

Structure of a Java Program

```
public class Student //class definition for a main class containing the main method
{
public static void main(String args[])
{
//statements
}
}
```

What is the main() method in Java?

- **Syntax of main() method:**

public static void main (String args[])

Explanations of the main() method

public- public access specifier allows the access of the method outside the program, since we want the JVM to identify the main method and start the execution from it, we want it to be marked **"public".**

static - The reason the main() method is marked static so that it can be invoked by JVM without the need of creating an object

void- The void means that the main() method will not return anything.

main () -This the default signature which is predefined by JVM. This makes the main method to be the entry point to the program

String args[] - The main method can also accepts string inputs that can be provided at the runtime. The inputs are stored in the array args[] of String type.

Java User input

- The Scanner class is used to get user input and its found in the java.util package.
- To use the Scanner class, create an object of the class and use any of the available methods found in the Scanner class documentation.
- The nextLine method is used to read strings and nextInt is used to read integers.

```
Scanner scanner= new Scanner(System.in);
```

Simple Java Program

Create a simple java program that accepts an ID and name of a student and then outputs it.

Lets solve this in netbeans IDE

Simple Java Program(Solution)

```
package student1;

/**
 *
 * @author Denokia
 */
import java.util.Scanner;
public class Student1 {

   /**
    * @param args the command line arguments
    */
   public static void main(String[] args) {
      // TODO code application logic here
      int ID; // declaring variable ID of integer type
      String name; // declaring variable name of  String type

      Scanner scanner=new Scanner(System.in); //

       System.out.println("Please enter your student name:");
       name=scanner.nextLine();


      System.out.println("Please enter your student_ID:");
      ID=scanner.nextInt();

      // display the student details

       System.out.println("Student_ID and name "+ " "+ID+"  "+name);

   }

}
```

Initializing an Object in Java

- Initializing an object means storing data into the object.
- There are three ways in which an object can be initialized in java and these are ;

- By reference variable
- By method
- By constructor

Initialization by Reference Variable

- Let's see a simple example where we are going to initialize the object through a reference variable.
- In the program below, we create two objects (std1 and std2) of type Student2 and the object fields (name, program and Student _ID) and objects are initialized with some values .

Initializing object through Reference variable

```
package student2;

/**
 *
 * @author Denokia
 */
public class Student2 {

    /**
     * @param args the command line arguments
     */
    String name,program; // declaring object field( name and program) of type String
    int student_ID; // declaring object field(student_ID) of type integer
    public static void main(String[] args) {
        // TODO code application logic here

        // creating two objects of type Student2
        Student2 std1=new Student2();
        Student2 std2=new Student2();

        // initialising first object
        std1.student_ID=202000025;
        std1.name="paul ssema";
        std1.program="ISM";

        // initialising second object
        std2.student_ID=202000026;
        std2.name="sarah Aguti";
        std2.program="ITM";

        System.out.println(std1.student_ID+" "+std1.name+" "+std1.program);
        System.out.println(std2.student_ID+" "+std2.name+" "+std2.program);

    }

}
```

Output

```
run:
202000025  paul ssema  ISM
202000026  sarah Aguti  ITM
BUILD SUCCESSFUL (total time: 1 second)
```

Initializing an object through Reference Variable

- We can also rewrite the above program by separating the main method from the object data .
- It's a good programming practice to create main class for the main method especially when dealing with big projects .
- The following program gives the same output

Initializing an object through Reference Variable

```
class TestStudent2 {

String name,program; // declaring object field( name and program) of type String
 int student_ID; // declaring object field(student_ID) of type integer

}


public class Student2 {

   /**
    * @param args the command line arguments
    */
public static void main(String[] args) {
      // TODO code application logic here

      // creating two objects of type TestStudent2
      TestStudent2 std1=new Student2();
      TestStudent2 std2=new Student2();

      // initialising first object
      std1.student_ID=202000025;
      std1.name="paul ssema";
      std1.program="ISM";

      // initialising second object
      std2.student_ID=202000026;
      std2.name="sarah Aguti";
      std2.program="ITM";

      System.out.println(std1.student_ID+" "+std1.name+" "+std1.program);
      System.out.println(std2.student_ID+" "+std2.name+" "+std2.program);

   }

}
```

Initializing object through method

- In this example, we are creating the two objects of Student3 class and initializing the value to these objects by invoking the insertRecord method. Here, we are displaying the state (data) of the objects by invoking the displayInformation() method.

```
package student3;

/**
 *
 * @author Denokia
 *
 *
 *
 */
class TestStudent3{
    // declaring the object fields
int rollno;
String name;
// defining the method for initialising value to the objects
void insertRecord(int r, String n){
 rollno=r;
 name=n;
}
// method for displaying the data of the objects
void displayInformation(){System.out.println(rollno+" "+name);}

}

public class Student3 {

  /**
   * @param args the command line arguments
   */
  public static void main(String[] args) {
     // TODO code application logic here

     // Creating two objects s1 and s2
 TestStudent3  s1=new TestStudent3();
 TestStudent3 s2=new TestStudent3();

 // invoking the insertRecord method to initialize value to the objects
 s1.insertRecord(100, "David");
 s2.insertRecord(101, "Samuel");

 // invoking the displayInformation method to display data stored in the objects
 s1.displayInformation();
 s2.displayInformation();

   }

}
```

Initializing object through constructor

- **This constructor simply initializes all fields of the object with their default values. Strings are initialized to *null* and integers to zero.**
- It is a special type of method which is used to initialize the object
- It is called when an instance of the class is created.
- **Note:** It is called constructor because it constructs the values at the time of object creation. It is not necessary to write a constructor for a class. It is because java compiler creates a default constructor if your class doesn't have any.

Initializing Object thru Constructor

Rules for creating Java constructor

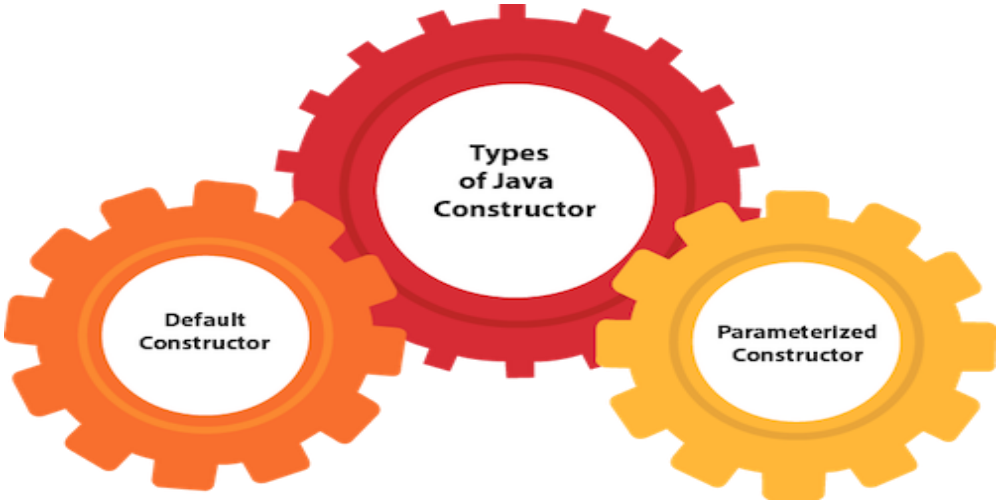
- There are two rules defined for the constructor.

1. Constructor name must be the same as its class name
2. A Constructor must have no explicit return type

- A Java constructor cannot be abstract, static, final, and synchronize

Initializing object through constructor

- Types of Java Constructors.



Initializing object through constructor

❑Java Default Constructor

- A constructor is called "Default Constructor" when it doesn't have any parameter.
- Syntax of default constructor:

<class_name>(){}

For example

Student3(){}

Example of default constructor that displays the default values

```
//Let us see an example of default constructor
//which displays the default values
class Student3{
int id;
String name;

// default constructor  initializes object fields to default values, int to 0 and String to null
Student3(){}

//method to display the value of id and name
void display(){System.out.println(id+" "+name);}

public static void main(String args[]){
//creating objects
Student3 s1=new Student3();
Student3 s2=new Student3();
//displaying values of the object
s1.display();
s2.display();
}
}
```

Output

o, null

o,null

Initializing object through constructor

Java Default Constructor (Example 2)

- In this example, we are creating the no-arg constructor in the Bike class. It will be invoked at the time of object creation.

```
//Java Program to create and call a default constructor
class Bike1{
//creating a default constructor
Bike1(){System.out.println("Bike is created");}
//main method
public static void main(String args[]){
//calling a default constructor
Bike1 b=new Bike1();
}
}
```

Initializing object through constructor

❑ Java Parameterized Constructor

The parameterized constructor is used to provide different values to distinct objects.

Example of parameterized constructor

- In this example, we have created the constructor of Student class that have two parameters.
- We can have any number of parameters in the constructor.

Initializing object through constructor

Java Parameterized Constructor (Example)

```
//Java Program to demonstrate the use of the parameterized constructor.
class Student4{
    int id;
    String name;
    //creating a parameterized constructor
    Student4(int i,String n){
    id = i;
    name = n;
    }

    //method to display the values
    void display(){System.out.println(id+" "+name);}

    public static void main(String args[]){

    //creating objects and passing values
    Student4 s1 = new Student4(111,"Karan");
    Student4 s2 = new Student4(222,"Aryan");

    //calling method to display the values of object
    s1.display();
    s2.display();
   }
}
```

Output

```
run:
111 Karan
222 Aryan
BUILD SUCCESSFUL (total time: 14 seconds)
```