Java OOPs Concepts

OOPs (Object-Oriented Programming System)

In OOPs concept we will learn about

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

As a part of above topic, we will learn about.

Java OOPs Concept

- Object Method, Class
- This Keywork
- Static Keyword
- Naming convention

Inheritance

- IS-A Relation
- HAS-A Relation

Polymorphism

- Method Overloading
- Method Overriding
- Return type
- IIB
- Runtime polymorphism
- Instance of Operator
- Final keyword

Abstraction

- Abstract class
- Polymorphism

Access Modifier
Object Class
Wrapper class
Math Class
Type Casting
Enum with Customized Value in Java

Object, Class and Method

Class
A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type.

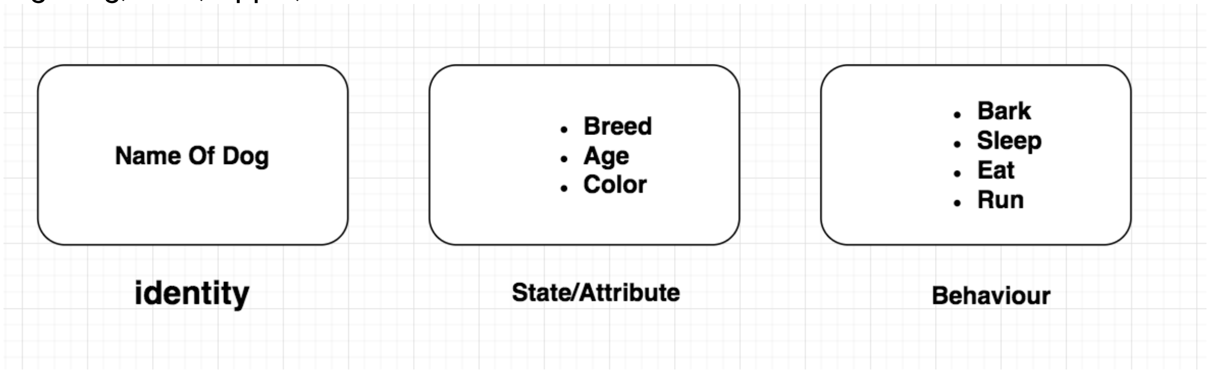
Class has

1. **Modifiers** : A class can be public or has default access.
2. **Class name:** The name should begin with a initial letter (capitalized by convention).
3. **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
4. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
5. **Body:** The class body surrounded by braces, { }.

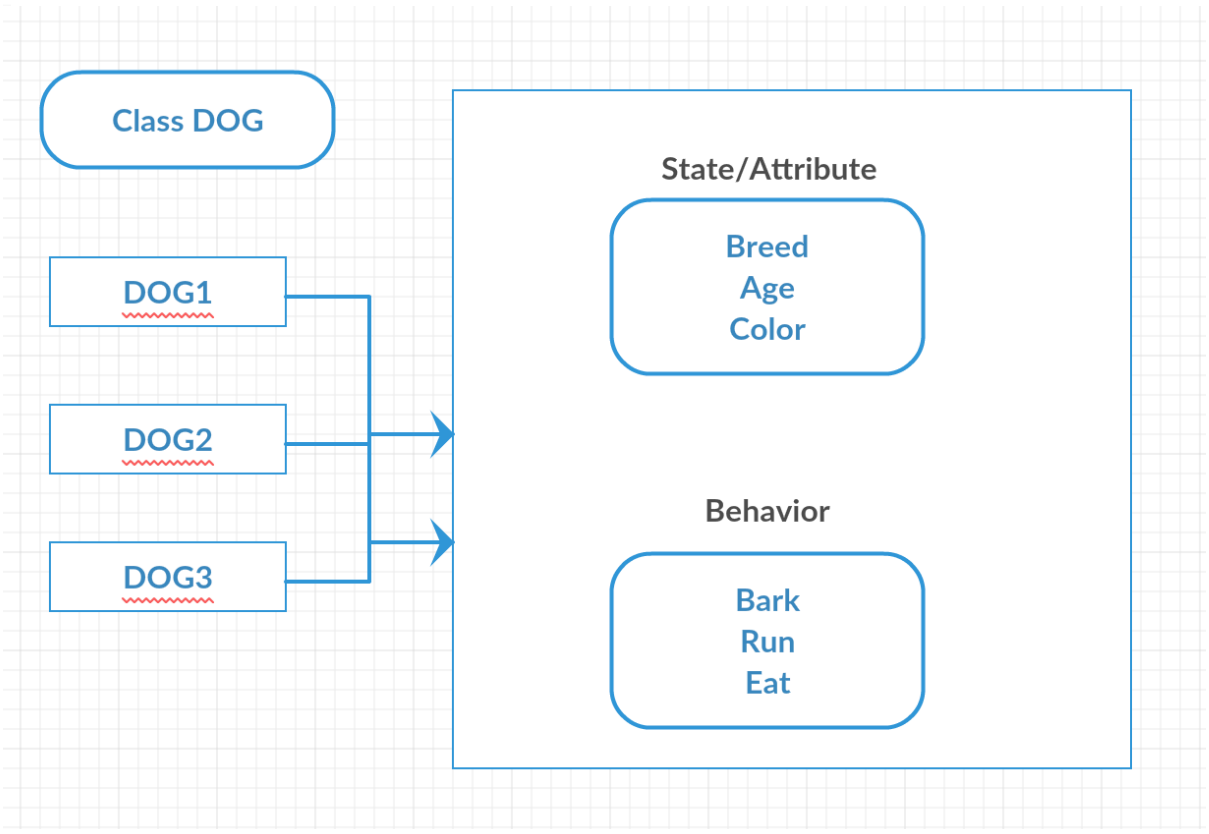
Objects correspond to things found in the real world.
An object consists of :

1. **State** : It is represented by attributes of an object. It also reflects the properties of an object.
2. **Behaviour** : It is represented by methods of an object. It also reflects the response of an object with other objects.
3. **Identity** : It gives a unique name to an object and enables one object to interact with other objects.

E.g. Dog, Cow, Apple, Ram



If we make 5 Object of same class, each object will have one copy of attributes/Behaviour



Method in Java

In Java, a method is like a function which is used to expose the behaviour of an object.

Advantage of Method

- Code reusability
- Code optimisation.

```
package ClassExample;

public class Example1 {
	int age;

	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}
	public static void main(String[] args) {
		Example1 obj1 = new Example1();
		obj1.setAge(4);
		obj1.getAge();
```

```
                Example1 obj2 = new Example1();
                obj1.setAge(6);
                obj2.getAge();
        }

}
```

public static void main(String[] args)

Java main method is the entry point of any java program. Its syntax is always public static void main(String[] args). You can only change the name of String array argument, for example you can change args to myStringArgs.

Also String array argument can be written as String... args or String args[].

public

This is the access modifier of the main method. It has to be public so that java runtime can execute this method. Remember that if you make any method non-public then it’s not allowed to be executed by any program, there are some access restrictions applied. So it means that the main method has to be public.

static

When java runtime starts, there is no object of the class present. That’s why the main method has to be static so that JVM can load the class into memory and call the main method. If the main method won’t be static, JVM would not be able to call it because there is no object of the class is present

void

Java programming mandates that every method provide the return type. Java main method doesn’t return anything, that’s why it’s return type is void. This has been done to keep things simple because once the main method is finished executing, java program terminates. So there is no point in returning anything, there is nothing that can be done for the returned object by JVM. If we try to return something from the main method, it will give compilation error as an unexpected return value.

```
package ClassExample;



public class Example1 {
        public static void main(String[] args) {

                return 0;

        }

}
```

String[] args

Java main method accepts a single argument of type String array. This is also called as java command line arguments.

Naming Convention in Java.

- Class name Should always start with Uppercase.
- Method should start with lower class.
- Package name should always be lowercase.
- Constant should be in uppercase.

```
package classExample;

public class Example1 {
        int age;
        static final int MAX_AGE = 18;

        public int getAge() {
                return age;
        }

        public void setAge(int age) {
                this.age = age;
        }

        public static void main(String[] args) {
                Example1 obj1 = new Example1();
                obj1.setAge(4);
                obj1.getAge();

                Example1 obj2 = new Example1();
                obj1.setAge(6);
                obj2.getAge();
        }
}
```

Constructors in Java

Constructors are used to initialize the object's state. Like methods, a constructor also contains **collection of statements(i.e. instructions)** that are executed at time of Object creation.

constructors are used to assign values to the class variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

When is a Constructor called ?
Each time an object is created using **new()** keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the **data members** of the same class.

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
3. A Java constructor cannot be abstract, static, final, and synchronized

Types of Java constructors
There are two types of constructors in Java:
- Default constructor (no-arg constructor)
- Parameterized constructor

Default Constructor

```java
package constructor;

public class Example1 {
	public String name;
	public int i;

	Example1() {
	}

	public static void main(String[] args) {
		Example1 obj = new Example1();
		System.out.println(obj.name);
		System.out.println(obj.i);
	}
}
```

Parameterized constructor

```
package constructor;

public class Book {

	int length;
	int breadth;
	int height;

	public Book(int length, int breadth, int height) {
		this.length = length;
		this.breadth = breadth;
		this.height = height;
	}

	public static void main(String[] args) {
		Book obj = new Book(10, 20, 30);
		System.out.println(obj.length);
		System.out.println(obj.breadth);
		System.out.println(obj.height);
	}
}
```

Key point to learn about constructor.

```
package constructor;

public class Exemple2 {
	int i ;
	public Exemple2(int i) {
		this.i = i;
	}

	public static void main(String[] args) {
		Exemple2 obj = Exemple2();
	}
}
```

```
package constructor;

public class Example3 {
	int i;
	public Example3(int i) {
		this.i = i;
		System.out.println("Parameterized");
	}

	Example3() {
   System.out.println("default");
	}

	public static void main(String[] args) {
		Example3 obj = new Example3();
		Example3 obj1 = new Example3(5);
	}
}


package constructor;

public class Example4 {

	private Example4(){
		System.out.println("default");
	}

	public static void main(String[] args) {
		Example4 obj = new Example4();
	}
}

package constructor;

public class Example5 {

	void Example5(){
		System.out.println("default");
	}

	public static void main(String[] args) {
		Example5 obj = new Example5();
	}
}
```

How constructors are different from methods in Java?

- Constructor(s) must have the same name as the class within which it defined while it is not necessary for the method in java.

- Constructor(s) do not return any type while method(s) have the return type or **void** if does not return any value.
- Constructor is called only once at the time of Object creation while method(s) can be called any numbers of time.

```
package constructor;

public class Example6 {

	Example6() {
		super();
	}

	public static void main(String[] args) {

	}

}
```

```
package constructor;

public class Example7 {

	Example7() {
		System.out.println("I am Example7()");
	}

	Example7(int i) {
		this();
		System.out.println("I am Example7(int i)");
	}

	public static void main(String[] args) {
		Example7 obj = new Example7(3);
	}
}
```

```
package constructor;

public class Example8 {
	Example8() {
		System.out.println("I am Example7()");
	}
```

```java
	Example8(int i) {
		this();
		System.out.println("I am Example7(int i)");
	}

	Example8(int i, int j) {
		this(5);
		System.out.println("I am Example8(int i,int j) ");
	}

	public static void main(String[] args) {
		Example8 obj = new Example8(3,6);
	}
}
```

```java
package constructor;

public class Example9 {

	private String name;
	private int age;
	private String state;

	public Example9(){

	}
	public Example9(String name, int age, String state) {
		this.name = name;
		this.age = age;
		this.state = state;
	}

	public void display(){
		System.out.println("Example9 [name=" + name + ", age=" + age + ", state=" +
state + "]");
	}

	public static void main(String[] args) {

		Example9 obj2 = new Example9();
		obj2.display();

		Example9 obj = new Example9("test", 23, "B");
		obj.display();

		Example9 obj1 = new Example9("test1", 24, "A");
		obj1.display();
	}
}
```

```
package constructor;

public class Example10 {

	private String name;
	private int age;
	private String state;

	public Example10(String name, int age, String state) {
		this.name = name;
		this.age = age;
		this.state = state;
	}

	public Example10(int age, String name, String state) {
		this.name = name;
		this.age = age;
		this.state = state;
	}

	public static void main(String[] args) {

	}

}
```

static keyword in java

static is a non-access modifier in Java which is applicable for the following:

1. blocks
2. variables
3. methods
4. nested classes

To create a static member(block,variable,method,nested class), precede its declaration with the keyword *static*. When a member is declared static, it can be accessed before any objects of its class are created, and without reference to any object.

```
package staticBlock;

public class Example1 {

	private static int a;

	public static void test1() {

	}

	static {
		System.out.println("this is sttaic block");
	}
}
```

Static Blocks

If you want to do some calculation in order to initialise the static variables, we can declare them in static block, and static block gets executed only once per class, when the class is first loaded