Classes and Objects

What is Class?

CLASS are a blueprint or a set of instructions to build a specific type of object. It is a basic concept of Object-Oriented Programming which revolves around real-life entities. Class in Java determines how an object will behave and what the object will contain.

Syntax

```
class <class_name>{
    field;
    method;
}
```

Example

```
class Dog {

    //declaring fields
    //fields are nothing but variables in class or struct
    String breed;
    String size;
    int age;
    String color;

    //declaring method
    void getInfo() {
        System.out.println("This is a Dog Class");
    }
}
```

What is Object?

OBJECT is an instance of a class. An object is nothing but a self-contained component which consists of methods and properties.

From a programming point of view, an object can include a data structure, a variable, or a function. It has a memory location allocated. The object is designed as class hierarchies.

Syntax

```
ClassName ReferenceVariable = new ClassName();
```

Example

```
//Consider Dog class given above
Dog d1 = new Dog();
```

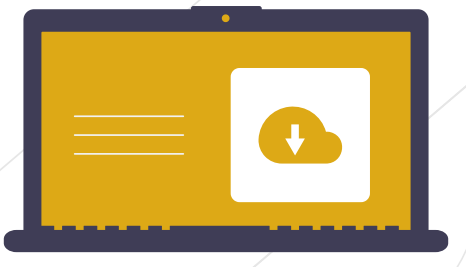
Still have a confusion in mind? Lets understand class and object with an example.

Let's take an example of developing a pet management system, specially meant for dogs. You will need various information about the dogs like different breeds of the dogs, the age, size, etc.

You need to model real-life beings, i.e, dogs into software entities. How do you design such software?

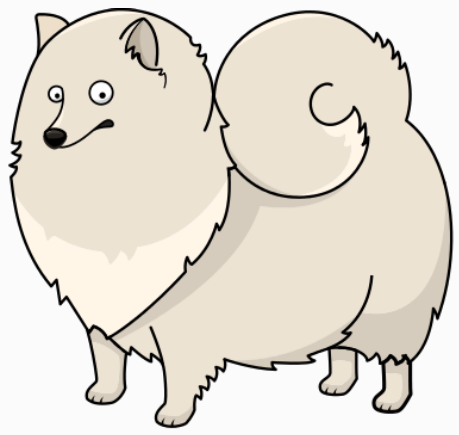
How to convert real life entities

dogs into software objects



First, let's do an exercise.
You can see the picture of three different breeds of dogs below.





Stop here right now! List down the differences between them.

Some of the differences you might have listed out may be breed, age, size, color, etc. If you think for a minute, these differences are also some common characteristics shared by these dogs. These characteristics (breed, age, size, color) can form data members for your object.

Common Characteristics

- [x] Breed
- [x] Age
- [x] Size
- [x] Color

Data members identified

Next, list out the common behaviors of these dogs like sleep, sit, eat, etc. So these will be the actions of our software objects.

Common Actioncs

- [x] Eat
- [x] Sit
- [x] Sleep
- [x] Run

Actions identified

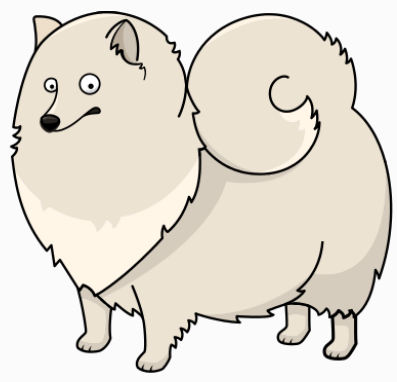
So far we have defined following things,

- **Class** - Dogs
- **Data members** - size, age, color, breed, etc.
- **Methods** - eat, sleep, sit and run.

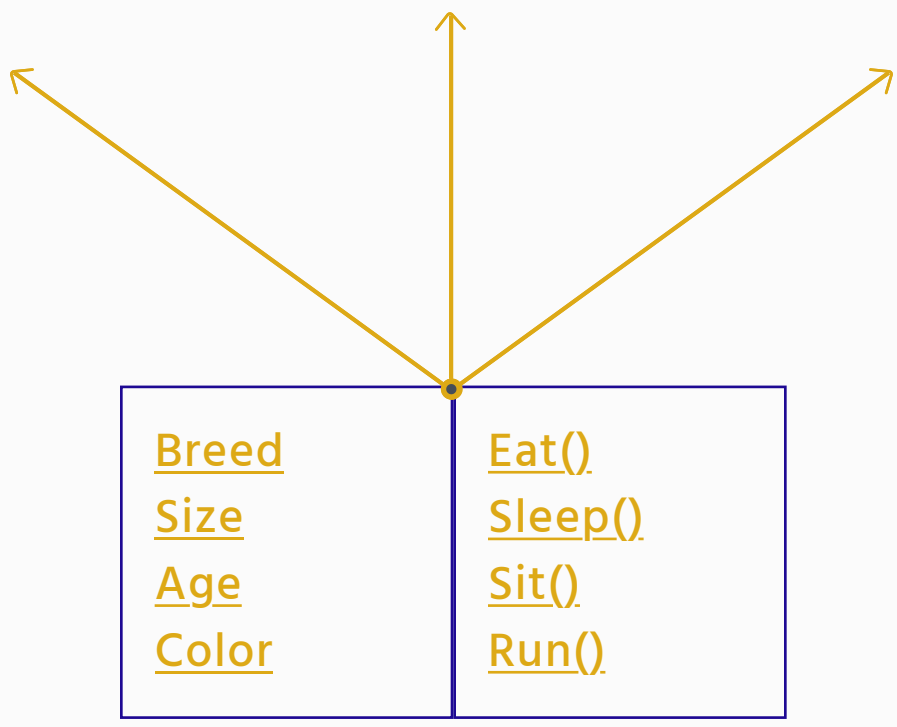
Now, for different values of data members (breed size, age, and color) in Java class, you will get different dog objects.

Breed = Neapolitan Mastiff
Size = Large
Age = 5 years
Color = Brown

Breed = Maltese
Size = Small
Age = 2 years
Color = White



Breed = Chow Chow
Size = Medium
Age = 3 years
Color = Off White



You can design any program using this OOPs approach.

Oh Great!!! Now you must have understood the concept of classes and objects.

Now let's take a look at how we can turn above knowledge into an actual JAVA program.

```java
// Class Declaration
public class Dog {
    // Instance Variables
    String breed;
    String size;
    int age;
    String color;

    // method 1
    public String getInfo() {
        return ("Breed is: "+breed+" Size is:"+size+" Age is:"+age+" color is: "+color);
    }

    public static void main(String[] args) {
        Dog maltese = new Dog();
        maltese.breed="Maltese";
        maltese.size="Small";
        maltese.age=2;
        maltese.color="white";
        System.out.println(maltese.getInfo());
    }
}
```

NOTE:

As Java is an object oriented programming language, every program of java has at least 1 class and a method in class called as **Main.**

Main method is an entry point for all java programs.

The name of the class in which the main method is declared must be the same as the name of java file.

Let's take further look on different ways to create object of a class

1. **Using new keyword** - This is the most common way to create an object in java. Almost 99% of objects are created in this way.

```
MyObject object = new MyObject();
```

2. Using Class.forName() - If we know the name of the class & if it has a public default constructor we can create an object in this way.

```
MyObject object = (MyObject) Class.forName("subin.rnd.MyObject").newInstance();
```

3. Using clone() - The clone() can be used to create a copy of an existing object.

```
MyObject anotherObject = new MyObject();
MyObject object = (MyObject) anotherObject.clone();
```

4. Using object deserialization - Object deserialization is nothing but creating an object from its serialized form.

```
ObjectInputStream inStream = new ObjectInputStream(anInputStream );
MyObject object = (MyObject) inStream.readObject();
```