enjoy algorithms

Search content...

Classes and Objects in Object Oriented Programming

To solve real-world problems using software, we'd like to define the real world's structure with attributes that exhibit the characteristics of the structure. So here, the idea of Class and Objects comes into the picture in object-oriented programming. In simple words, classes and objects are the fundamental building blocks of Object-Oriented Programming, which help us to implement several key concepts in OOPS.

What is a Class?

A class is a user-defined layout or blueprint of an object that describes what a specific kind of object will look like. A class description consists of two things: 1) Attributes or member variables, and 2) Implementations of behavior or member functions.

So in object-oriented terminology: A class is a blueprint that defines the variables and the methods common to all objects of a certain kind. It helps us to bind data and methods together, making the code reusable, unlike procedural language.

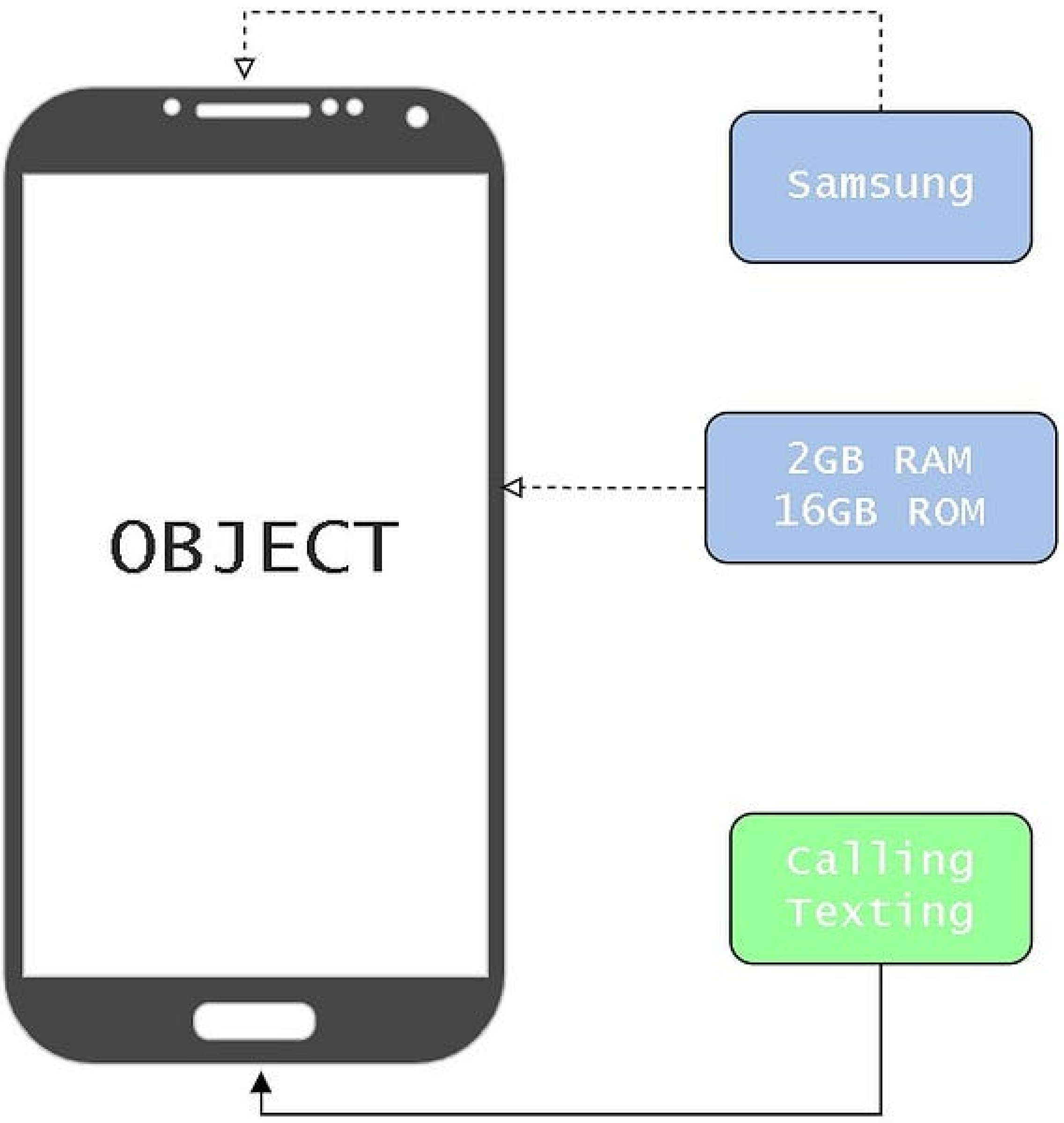
For example, a **mobile phone** has attributes like a brand name, RAM, and functions like texting and calling. Thus, the mobile phone is a class of various phones (the objects).

What is an Object?

An object is a single instance of a class, which contains data and methods working on that data. So an object consists of three things:

- Name: This is a variable name that represents the object.
- Member data: The data that describes the object.
- Member methods: Behavior that describes the object.

For example, **Samsung Galaxy** is an object with the brand name Samsung, 2GB RAM as properties, and calling and texting as behaviors.



The mobile phone class can be defined as:

```
public class MobilePhone {
      //attributes
        private String imeiNumber;
        public String name;
        public int RAM;
      //methods
        public String getName() {
            return name;
        }
        public int getRAM() {
            return RAM;
        }
}
```

Special Notes

When a class is defined, no memory is allocated, but memory is allocated when an object is created.

We can create many objects from the same class type.

An object is a concrete ‘thing’ that we make using a specific class and the word ‘instance’ indicates the relationship of an object to its class. For example, suppose we have a class Phone. Then android phones and iPhones are instances of the class Phone.

Software programmers use the same class, and thus the same code, over and over again to create many objects. So objects provide the benefit of modularity and information hiding and classes provide the benefit of reusability.

Components of a Class

Access modifier: A class can be made public by adding the keyword *public* before its name. A public class can be accessed by any class of any module, whereas a default class is accessible within the same code only.

The *class* keyword is then followed by the **class name** (MobilePhone), which must be a valid identifier.

The body of the class contains the declaration of properties, also known as **instance variables,** and class methods (behaviors).

Each method/attribute is preceded by an **access specifier** *private, protected, or public,* where private is the default specifier. We’ll learn more about specifiers when we’ll deal with objects.

Note: The attributes can be any valid data type, either user-defined (a class is also a user-defined data type) or primitive. The methods are the general functions.

Class Declaration: A class declaration shows what an object will look like and what its available functions are. This gives an interface (what the user sees), where the user of this interface does not need to know the implementation details. The user of a class uses the class by creating objects and calling the available methods for those objects.

Class Definition: This is the class implementation details, which consist of definitions of its members. It is invisible to the user of the interface.

Declaring the objects

Recall that **int** is a data type, and to declare an int type variable, we follow the syntax — int variable_name. Similarly, a class is a user-defined data type; therefore, to create an **object**, we create variables that hold the objects.

```
MobilePhone iPhone;
MobilePhone Nord;
```

In java, the variables (iPhone and Nord) that point to or hold the object are the **reference variables**. iPhone and Nord are currently holding **null** values(default values). To store actual objects in these variables, we need to initialize them with the objects.

Initializing and using the objects

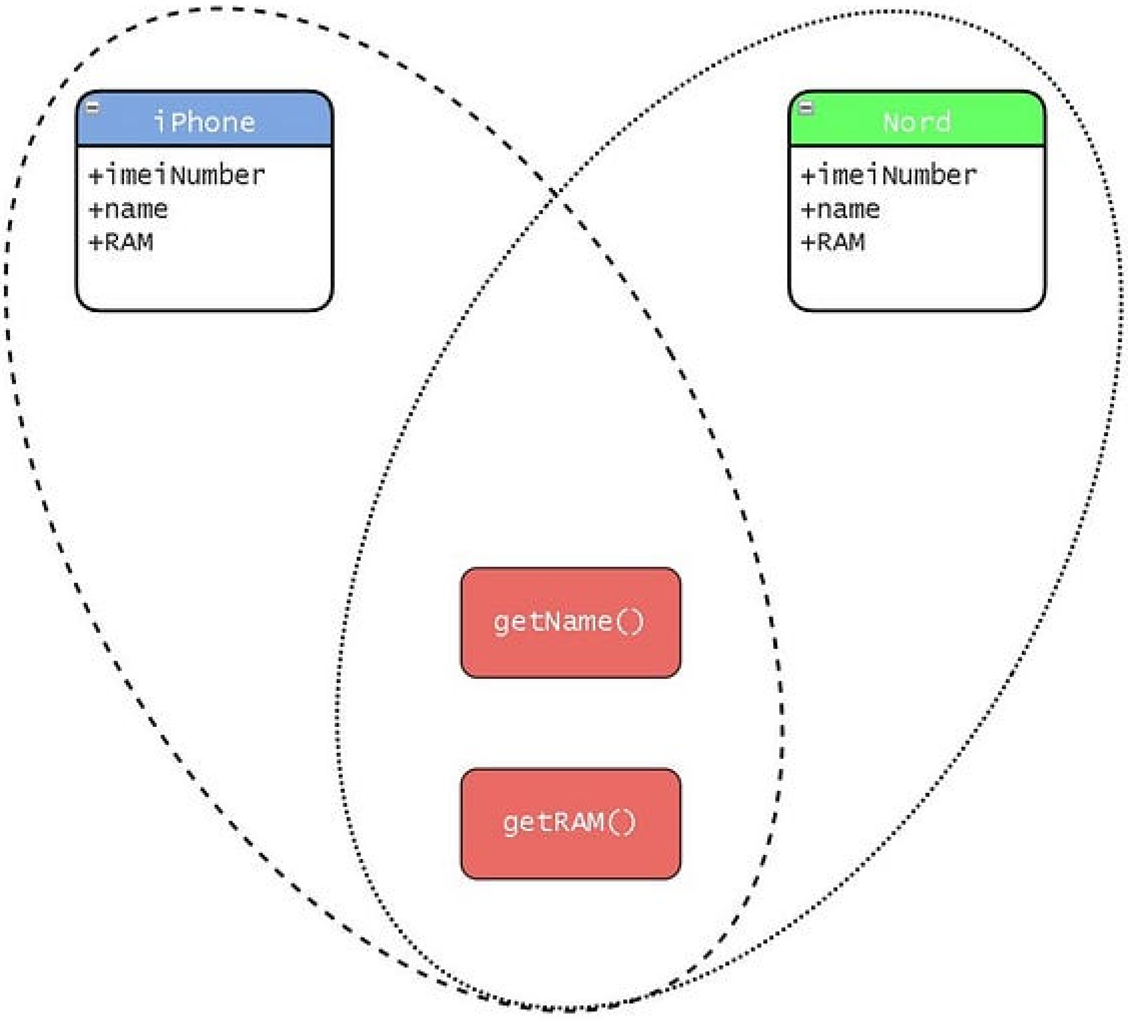
To initialize an object, we generally use the *new* operator:

```
Syntax:
<variable name> = new <Class name>();

Examples:
iPhone = new MobilePhone();
Nord = new MobilePhone();
```

In the above code, MobilePhone() refers to the **constructor** of MobilePhone class. A constructor is a special method that contains the code that runs when we instantiate an object. In java, the **new operator** allocates memory in the heap for a new object and returns a reference to this memory.

Each instance of a class has its own set of instance variables. But, all instances have a reference to a common set of methods declared in the class. Here’s a visual representation of the same:



We can access different attributes and methods of instantiated objects using the **dot operator**. For example, to use the getName() method of iPhone, we write iPhone.getName().

Importance of access specifiers

Try this operation:

```
System.out.println(iPhone.imeiNumber);
```

This is invalid because *imeiNumber* is a private instance variable. By definition, any class instance cannot directly access private or protected data members (attributes and methods). In other words, classes implement **data hiding** by declaring sensitive data members as private/protected and the rest as public.

Use case of classes and objects in OOPs

With classes and objects, we can perform data hiding and inheritance.

Encapsulation is packing data and related operations in a single unit. Classes help in encapsulating state and behavior.

Abstraction is a process of hiding the internal details of an application from the outer world and providing only meaningful details. In java, we implement abstraction with abstract classes and interfaces.

Some special classes exist, especially in java, such as abstract class, interface, and concrete class.

Conclusion

Classes and objects are very instrumental in depicting real-life entities. They are used in implementing various OOPs concepts and provide the basis of OOPs.

Additional Reading

To know more about OOPs, one can refer to our existing blogs

Fundamentals of OOPS in Java

Fundamentals of OOPS in C++

Enjoy learning, Enjoy oops!

Author: Ankit Nishad

Reviewer: Mahendra Chouhan

Share:

Related Tags: oops-concepts oops-basics

Share Feedback

Name

Email

Message

Submit Your Response

Coding Interview

DSA Course

DSA Blogs

Machine Learning

ML Course

ML Blogs

System Design

SD Course

SD Blogs

OOP Concepts

OOP Course

OOP Blogs

EnjoyAlgorithms Newsletter

Subscribe to get well designed content on data structure and algorithms, machine learning, system design, object orientd programming and math.

Email address

Subscribe

Explore More Content

Courses	Tags	Latest Blogs
About Us	Contact Us	Stories

Follow us on