Java - What is OOP?

OOP stands for **Object-Oriented Programming**.

object-oriented programming is about creating objects that contain both data and methods.

Object-oriented programming has several advantages over procedural programming:

* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the Java code DRY "Don't Repeat Yourself", and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

**Object** − Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors – wagging the tail, barking, eating. An object is an instance of a class.

**Class** − A class can be defined as a template/blueprint that describes the behavior/state that the object of its type support

If a class has multiple methods having same name but different in parameters, it is known as **Method Overloading**

Advantage of method overloading

Method overloading *increases the readability of the program*.

**Different ways to overload the method**

There are two ways to overload the method in java

* By changing number of arguments
* By changing the data type

Method overriding is declaring method in subclass which is already present in parent class.

Constructor = is special method that is called when an object is created (instantiated)

Polymorphism means "**many forms**", and it occurs when we have many classes that are related to each other by inheritance.

Inheritance lets us inherit attributes and methods from another class. Polymorphism uses those methods to perform different task.

For example, think of a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs, Birds - And they also have their own implementation of an animal sound (the pig oinks, and the cat meows, etc.):

The **super** keyword in Java is a reference (link) variable which is used to refer immediate parent class object.

Encapsulation

* **Encapsulation in Java** = attributes of a class will be hidden or private,

**Can be accessed only through methods (getter and setter)**

**U should make attributes private if u don’t have a reason to make them public or protected**

* variables/attributes as private
* provide public **get** and **set** methods to access and update the value of a private variable

Why Encapsulation?

* Better control of class attributes and methods
* Class attributes can be made **read-only** (if you only use the get method), or **write-only** (if you only use the set method)
* Flexible: the programmer can change one part of the code without affecting other parts
* Increased security of data

Access Modifiers

* **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
* **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
* **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
* **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package

Non-Access Modifiers

Final the final keyword is a non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override).

|  |  |
| --- | --- |
| final | Attributes and methods cannot be overridden/modified |

Static a static method means that it can be accessed without creating an object of the class

|  |  |
| --- | --- |
| static | Attributes and methods belongs to the class, rather than an object |

Abstract

The abstract keyword is a non-access modifier, used for classes and methods.

**Class:** An abstract class is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).

**Method:** An abstract method can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

**Interfaces** : Another way to achieve abstraction in Java, is with interfaces.

An interface is a completely "abstract class" that is used to group related methods with empty bodies:

**Difference between Enums and Classes**

An enum can, just like a class, have attributes and methods. The only difference is that enum constants are public, static and final (unchangeable - cannot be overridden).

An enum cannot be used to create objects, and it cannot extend other classes (but it can implement interfaces).

Why And When To Use Enums?

Use enums when you have values that you know aren't going to change, like month days, days, colors, deck of cards, etc.

Inheritance is a mechanism of driving a new class from an existing class. The existing (old) class is known as parent class. The new class is known as a child class. It allows us to use the properties and behavior of one class (parent) in another class (child).

Type of inheritance:

Single Inheritance

Multi-level Inheritance

Hierarchical Inheritance

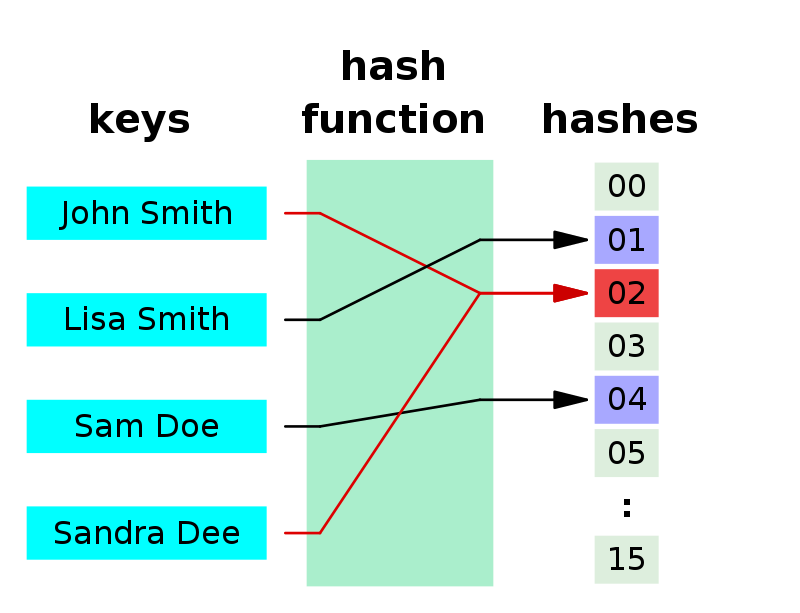
Hybrid Inheritance

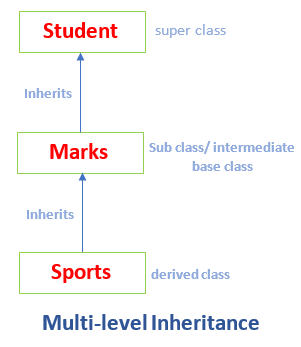
**Hash table**,

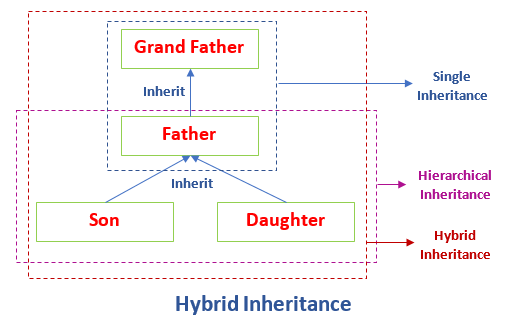
Also known as **hash map**, is a [data structure](https://en.wikipedia.org/wiki/Data_structure) that implements an [associative array](https://en.wikipedia.org/wiki/Associative_array) or dictionary. It is an [abstract data type](https://en.wikipedia.org/wiki/Abstract_data_type) that maps [keys](https://en.wikipedia.org/wiki/Unique_key) to [values](https://en.wikipedia.org/wiki/Value_(computer_science)).[[2]](https://en.wikipedia.org/wiki/Hash_table#cite_note-ms-2) A hash table uses a [hash function](https://en.wikipedia.org/wiki/Hash_function) to compute an *index* (*hash code)*

Hash collision

In a **hash collision** or **hash clash** is when two pieces of data in a [hash table](https://en.wikipedia.org/wiki/Hash_table) share the same hash value. The hash value in this case is derived from a [hash function](https://en.wikipedia.org/wiki/Hash_function) which takes a data input and returns a fixed length of bits







Object class in Java

The Object class is the parent class of all the classes in java by default. In other words, it is the topmost class of java.

The Object class is beneficial if you want to refer any object whose type you don't know. Notice that parent class reference variable can refer the child class object, know as upcasting.

Let's take an example, there is getObject() method that returns an object but it can be of any type like Employee,Student etc, we can use Object class reference to refer that object. For example:

Covariant return types in Java

Covariant return type refers to return type of an overriding method. It allows to narrow down return type of an overridden method without any need to cast the type or check the return type. Covariant return type works only for non-primitive return types

Java Lambda Expressions

A lambda expression is a short block of code which takes in parameters and returns a value. Lambda expressions are similar to methods, but they do not need a name and they can be implemented right in the body of a method.

1. () -> {
2. //Body of no parameter lambda
3. }
4. (p1) -> {
5. //Body of single parameter lambda
6. }
7. **interface** Drawable{
8. **public** **void** draw();
9. }
10. **public** **class** LambdaExpressionExample {
11. **public** **static** **void** main(String[] args) {
12. **int** width=10;
14. //without lambda, Drawable implementation using anonymous class
15. Drawable d=**new** Drawable(){
16. **public** **void** draw(){System.out.println("Drawing "+width);}
17. };
18. d.draw();
19. }
20. }
21. **interface** Drawable{
22. **public** **void** draw();
23. }
25. **public** **class** LambdaExpressionExample2 {
26. **public** **static** **void** main(String[] args) {
27. **int** width=10;
29. //with lambda
30. Drawable d2=()->{
31. System.out.println("Drawing "+width);
32. };
33. d2.draw();
34. }
35. }

The **double colon (::) operator** in Java, is used to call a method by referring to it with the help of its class directly

**Syntax:**

<Class name>**::**<method name>

6489