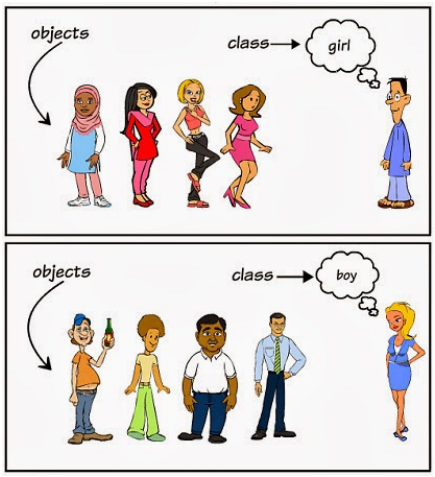
OBJECT AND CLASSES

Classes- Introduction

A class is a template for an object, and an object is an instance of a class. A class creates a new data type that can be used to create objects. It is a logical entity. It can't be physical. Classes can be inbuilt or user-defined.



A class in Java can contain:

- Fields
- Methods
- Constructors
- Blocks
- Nested Class and Interface

For example dogs have states or data members like colour, breed and the behaviour or methods (actions it can perform) are barking, wagging tail, eating, sleeping, etc.

Classes are of generally four types:

- Derived Classes
- Super Classes
- Sub Classes
- Mixed Classes

Derived Classes

A derived class has been generated or derived from another class. Its purpose is to improve the functionality of the base class. This sort of class derives properties from other classes and inherits them. Example: Let's take an example of a child and parents, a child has all the properties of a father. To be discussed later.

Sub Classes

A class that inherits its properties from another class is referred to as a subclass. The class from which properties are inherited is called the superclass. To be discussed later.

Super Classes

A superclass is a class from which numerous subclasses can be derived. Animal is the superclass and Bird, Dog, and Fish are the subclasses. To be discussed later.

Objects- Introduction

An entity that has state and behaviour is known as an object e.g., chair, bike, marker, pen, table, car, etc. It can be physical or logical (tangible and intangible).

An object has three characteristics:

- State: The state of an object is a value from its data type.
- Behaviour: The behaviour of an object is the effect of data-type operations.
- Identity: The identity of an object distinguishes one object from another. It is useful to think of an object's identity as the place where its value is stored in memory.

When we declare an object of a class, you are creating an instance of that class. Thus, a class is a logical construct. An object has physical reality, i.e. an object occupies space in memory.

Lifecycle of an Object

There are four parts in the lifecycle of an Object:

- Object Creation
- Object Accessing
- Object Unreferred
- Garbage Collection

Object Creation

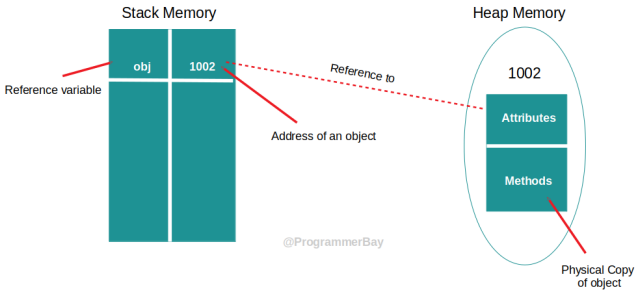
The object is a basic building block of an OOP's language. In Java, we cannot execute any program without creating an object. There is various way to create an object in Java that we will discuss in this section, and also learn how to create an object in Java.

Java provides five ways to create an object.

- Using new Keyword.
- Using clone() method.
- Using newInstance() method of the Class class.
- Using newInstance() method of the Constructor class.
- Using Deserialization.

New Keyword

Object is created using **new** keyword in the heap memory. The new keyword dynamically allocates memory for an object & returns a reference to it. This reference is, more or less, the address in memory of the object allocated by new. This reference is then stored in the variable. Thus, in Java, all class objects must be dynamically allocated.



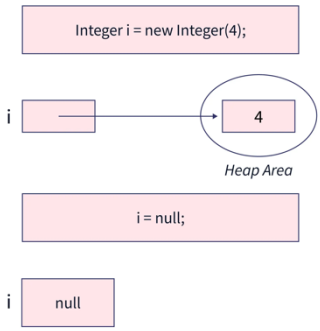
The object is created in the heap memory whereas the reference as value of variable assigned is stored in the stack memory. From the stack its pointing toward the created object in heap. All the attributes are stored inside the object only if they instance. The static variables were stored in a Java Memory Section known as Method Area, but after Java 8 they are also stored inside heap memory, but not inside the object.

All the variables are initialised inside the object with given values if not then the default values by the constructor of the class(to be discussed later in detail in Constructor chapter).

Other ways of creating objects will be discussed in Exception Handling Chapter.

Object Accessing

Instance variables are accessing through objects using dot(.) operator. **Once we have created an object, we probably want to use it for something.** We may need to use the value of one of its fields, change one of its fields, or call one of its methods to perform an action.



Object Unreferred

When an object inside gets unreferred this it becomes inaccessible. The Garbage Collector comes into action then.

Garbage Collection

In Java applications, objects are stored as per the requirement. The garbage collector (GC) task in JVM is to determine which memory is not being used by the application automatically and to recycle the memory for other tasks. Since this happens automatically with the help of JVM, developers need not have to do it to free up the memory space explicitly. Garbage collection frees the programmer from manually cleaning up the space and dealing with memory deallocation.

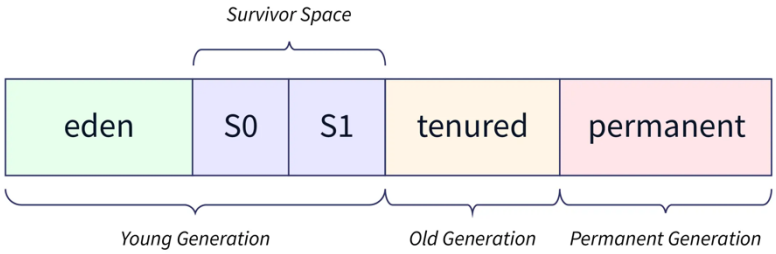
Usually, the garbage collection in Java is of two types:

- **A minor or incremental Java garbage collection**: It has occurred when unreachable objects in heap memory are removed.
- **A major or full Java garbage collection:** It occurs when objects that have survived the minor garbage collection are removed from heap memory.

In order to free up the memory space, the JVM must stop the application for a short interval of time in order to execute GC. It means that all the threads, except GC, will stop executing until objects are freed up by the garbage collector.

In modern java implementation of GC is such that most of the work happens in the background while the keep running the application. For this, a separate thread is used. For example, in order to clear the space used by an unreachable object, a separate thread is allotted while in the background, the application is running. GC in JVM consumes a lot of CPU resources for freeing up the memory space. There are four ways in which the activity of GC is performed:

- Serial- In this, only one thread gets executed by GC.
- Parallel(Default)- In this, multiple threads and minor threads get executed simultaneously each executing a part of GC.
- Concurrent Mark Sweep (CMS)- It is similar to that of parallel except for the fact that it allows execution of some threads of an application while reducing the frequency of other threads. But it may stops the function of application threads. No Compaction also.
- G1- It also runs parallelly and concurrently but has functions different from that of CMS. It guarantees that it will not stop the function of application thread while running GC. Also provides compaction.



There are 3 types of the heap where GC takes place:

- Young Generation: The newly created objects reside in this. It is then divided into two:
  - Eden space- The newly created objects are stored in Eden Space.
  - Two survivor spaces- When Mark and Sweep Algorithm runs for the first time it sends the survived objects from Eden to S0 with age equals 1 and deletes the unreferred objects. For the second time when Mark and Sweep Algorithm runs it sends the objects from Eden to S1 with age equals 1 and S0 to S1 with age 2. When the Algorithms runs for the third time it again sends the objects from Eden to S0 with age equals 1, S1 to S0 with age 2 and 3. The objects having age equals 3 got there threshold and hence they are sent to Old Generation. Whatever performs in Young Generation of deletion is called Minor GC.
- Old Generation: Here the objects that are in the heap for a long time are termed as old generation. Objects are moved from young to old. When an object is collected from the old generation, it is called a Major GC.
- Permanent Generation: In this, classes and methods are stored and collected that are final. Before it was called PermGen but now it's known as Meta Space.

There are 4 types of References in Heap:

- Strong References: This is the default type/class of Reference Object. Any object which has an active strong reference are not eligible for garbage collection. The object is garbage collected only when the variable which was strongly referenced points to null.

- **Weak References:** Weak Reference Objects are not the default type/class of Reference Object and they should be explicitly specified while using them.
  - This type of reference is used in WeakHashMap to reference the entry objects .
  - If JVM detects an object with only weak references (i.e. no strong or soft references linked to any object object), this object will be marked for garbage collection.
  - To create such references java.lang.ref.WeakReference class is used.
  - These references are used in real time applications while establishing a DBConnection which might be cleaned up by Garbage Collector when the application using the database gets closed.

- **Soft References:** Soft References: In Soft reference, even if the object is free for garbage collection then also its not garbage collected, until JVM is in need of memory badly. The objects gets cleared from the memory when JVM runs out of memory. To create such references java.lang.ref.SoftReference class is used.
- **Phantom Reference:** The objects which are being referenced by phantom references are eligible for garbage collection. But, before removing them from the memory, JVM puts them in a queue called 'reference queue' . They are put in a reference queue after calling finalize() method on them. To create such references java.lang.ref.PhantomReference class is used.

The garbage collection is important in order to free up the space and resources and to let other objects use that space instead of creating a new one. It helps minimize the memory used in the heap and helps the program to run more efficiently.