```
Class
=====
A class is a set of common properties and methods that are common to all
objects of one type.

Ex: Employee

What all properties can be common to all employees?

All employees work for some [company] and gets a [basic salary] for
his/her [work].

The company maintains a record of all employees' [names] and assigns each
of them an [employee id].

An employee [is permanent] once he/she lives through the probation
period.

The company can [see the details] of the employee whenever needed.

The company can [calculate total salary] for all the employees by
evaluating basic salary, standard deductions and additions.

So, properties of every employee can be
- employee id
- name
- basic salary
- is permanent employee
- company name

And common process for every employee will be
- display()
- calculateTotalSalary()

Syntax:

class Employee {
      // properties
      int empId;
      String name;
      double salary;
      String companyName;
      boolean isPermanent;

      // method
      display();
      calculateTotalSalary();
}

Object
======
Now that we have created a set of common properties, its time to create
real world entities out of them
```

```
Ex 1: Lets create a Genpact employee with name "Yash" who gets a basic
salary of 35000 and is assigned an emp id 101.

Employee employee1 = new Employee();
employee1.empId = 101;
employee1.name = "Yash";
employee1.salary = 35000;
employee1.company = "Genpact";
employee1.isPermanent = true;
employee1.calculateTotalSalary();

Ex 2: Create another temporary Genpact employee with name "Raj" who gets
a basic salary of 15000 and is assigned an emp id 102.

Employee employee2 = new Employee();

Yash and Raj are real Employees and so, these are objects of type
Employee
-----X-----X-----
Reference Variable
==================
In the above example, employee1 and employee2 are known as reference
variables.
They store the address of actual object created by new operator.

new operator
============
- new is a dynamic memory allocation operator.
- It is used to allocate memory to an object at runtime.
- These objects are created in heap memory.
- It creates the object and returns its memory address.
- It invokes the constructor of the class to initialize data members

Instance
========
- Instance is just another name for object.
- Whatever is created by new is the object/instance.
- For ex: new Employee() is the object (aka Instance)

Note:
- Only when new is written instance/object is created
- Only when new is written, instantiation is done

Instantiation
-------------
Creating an object of a class with the new operator is known as
Instantiation.

Type reference-name = new Type-constructor(parameters-list)
Employee employee1 = new Employee();
Employee employee2 = new Employee("Raj", 25000);

Instance Variables
==================
- Do not confuse instance with instance variables
- In the example: Employee employee1 = new Employee()
      - employee1 is a reference variable
      - new Employee() is the object/instance
      - empId, name, salary, isPermanent are instance variables of
Employee class
      - Instance variables are created for the object/instance
- Instance variables are also known as data members, attributes,
properties, state, fields etc

Class Members
=============
The things that we can write inside a class are called class members. A
class can have:
1. Instance variables
2. Static variables
3. Constructors
4. Methods
5. Inner classes
6. Member interfaces
7. Blocks

*Variables, constructors and methods are the most important building
blocks of a class.
-----X-----X-----X-----
Methods
=======
Methods are used to perform a task or group of tasks.
These tasks can be:
1. Setting data in an instance variable.
2. Getting data of an instance variable.
3. Displaying data
4. Process (filter, sort) the data
5. Pass data to other processing units etc

Methods have the syntax:

<access-modifier(s)> <return-type> methodName(<params-list>)
{
      method body
}

Examples:

1. Method without return statement without parameter : void display(){}

2. Method without return statement but with parameter: void
display(String type){}

3. Method with return statement without parameter: double
calculateTotalSalary(){}

4. Method with return statement with parameter: double
calculateTotalSalary(double deductions, double additions){}
```

```
Examples of methods:
1. void display(){}
2. void display(String type){}
3. double calculateTotalSalary(){}
4. double calculateTotalSalary(double additions){}

Constructor
===========
A constructor is used to initialize the data members of an object.

There are 2 types of constructors:
1. Default Constructor
2. Parameterized Constructor

Default constructor
-------------------
- Employee() is default constructor
- It is also known as Non-parameterized constructor
- It initializes the data members by default values
      -> 0 for byte, short, int, long.
      -> 0.0 for float, double
      -> false for boolean
      -> null for String
      -> '\u000' for char

- If we don't define our own default constructor, java provides it
automatically/implicitly.

- If we define our own parameterized constructor, java stops providing
the default constructor.

Rules to define a constructor
-----------------------------
- The name of the constructor should be same as the name of the class
- A constructor is very similar to a method, i.e. it can have parameters-
list and has a body.
- A constructor does not have a return type

Parameterized Constructor
-------------------------
Employee(int empId, String name, double salary){}

The above is parameterized constructors
- We can define as many parameterized constructors as we want

Instance Variable Hiding
========================
- If local variables and instance variables have the same name within a
method/constructor, local variables will hide instance variables

Ex:
class Employee {
      // empId is instance variable
```

```
	int empId;

	// empId is local variable
	Employee(int empId){
		// local variable hides instance variable
		empId = empId;
	}
}
```

```
this
====
- Inorder to distinguish between local and instance variables, we use
this keyword

- this represents the calling object

Ex:

Employee(int empId){
	//this.empId is now instance variable
	this.empId = empId;
}

Signature
=========
The name of the method/constructor along with the number, type and
sequence of parameters is known as signature.

ex: Following are signatures:
- display()
- display(String)
- Employee()
- Employee(int, String, double)
- Employee(String, double)

Overloading
===========
When the names of 2 methods/constructors are same but their signatures
are different it is known as Overloading.

Overloading can be achieved for both: methods and constructors

Constructor Overloading
-----------------------
When overloading is done for constructors, it is known as Constructor
overloading.

In class Employee:

Employee()
Employee(String name, double salary)
Employee(int empId, String name, double salary)

Names are same but signatures differ.
```

```
Method Overloading
------------------
When overloading is done for methods, it is known as Method overloading

display()
display(String type)

names are same but signatures are different.

Polymorphism
============
Polymorphism means same name different functionality

There are 2 types of Polymorphism:
1. Compile Time Polymorphism
2. Run Time Polymorphism

Compile Time Polymorphism
-------------------------
- In Java, Compile Time Polymorphism is achieved through
method/constructor overloading.
- Here the call to the appropriate method is resolved at compile time
- It is also known as
      - Static binding
      - Early binding
      - Eager binding
      - Static Polymorphism

Run Time Polymorphism
---------------------
- In Java, Run Time Polymorphism is achieved through method overriding.
- We will learn about it in the topic Inheritance

-----X-----X-----X-----
Variables
=========
- Variables are used to hold data
- It is mandatory to declare a variable before using it
- Variable is visible only in scope where it is declared
- There are 3 types of variables: instance, static and local variables.
- Local variables are defined inside a method/constructor.
- Variables should be named in lowerCamelCase.
Ex: simpleInterest, noOfYears etc
- A particular type of variable can hold only that type of data. What it
means is : an integer variable can hold only integer value.
- We cannot declare a variable by the same name within the same scope

There are 3 types of variables:
1. Local variables
2. Instance variables
3. Static Variables

Local Variables
```

```
---------------
The variables defined inside a method or constructor are called local
variables.

In the constructor:
Employee(int e, String n, double s){

}
e, n, s are local variables

In the method:
void display(String type) {

}

type is a local variable

In the method:

double calculateSalary (double additions){

      double total = basicSalary + additions;
      return total;
}

total and additions ar local variables.

- Local variables must be initialized before using, but paramters are not
initialized.
- Parameters will get their values during method call
- The scope of Local variables is limited to its method/constructor

Instance Variables
------------------
- Variables declared outside any method or constructor are instance
variables.
- They are created for the use of instance.
- For every instance a separate copy of instance variables are craeted in
memory
- Instance variables are initialized by constructors

Static variables
----------------
- Variables declared outside any method or constructor and preceded by
keyword static are static variables.
- They are created for the use of class.
- Only a single copy of static variable is created in memory
- Static variables are initialized once when the 1st time constructor is
called
- This single copy is shared by all instances and class
Ex: companyName, material, totalSurfaceAreas

final variables
---------------
```

```
- In order to create a constant we use final keyword
- The final keyword can be used with variable, method and a class
- The value inside a variable declared as final cannot be changed/re-
assigned

Static Method
=============
Methods are declared static when
- we don't want to create an object of a class to call it
- it has to work on static data
- it defines a fucntionality that is not specific to one object

Features of "static"
--------------------
- static is related with [class] [not with objects]
- static variables/methods are known as a class variables/methods
- never ever use a static variable/method with an object
- Sub classes can inherit static variables
- Sub classes can inherit static methods
- Sub classes cannot override a static method
- With static methods, overriding is not possible but overloading can be
done
- we cannot use static with parameters or local variables

Encapsulation
=============
Putting/Bundling all the data members, constructors and member methods
inside a class is known as Encapsulation

Data Hiding/Protection
----------------------
- To secure data members, we restrict their access with access modifiers.
- This is known as Data Hiding/Protection

Setter and Getter
=================
Setter - Modify object properties
---------------------------------
Constructor is used to create an object
Employee employee1 = new Employee("Yash", 20000);

Setter method is used to modify the value of a property
employee1.setSalary(30000);

The salary will change from 20000 to 30000

Getter - Access object properties
---------------------------------
- Usually the data members are private.
- Hence they cannot be accessed outside their own class.

If we want to access them outside their own class, we create getter
methods.
double salary = employee1.getSalary();
```

```
static v/s final v/s private
============================
static variable : Only one copy will be created and we can change the
value of the variable

final static : Only one copy will be created and we can not change the
value of the variable but we can use it outside the class

private final static : Only one copy will be created and we can not
change the value of the variable and we can not use it outside the class
-----X-----X-----
null
====
- null acts as a default value for reference types (like objects, arrays,
String etc)
- null means that the object reference is initialized
- The object reference does not point to any object.
- It indicates the absence of a value or object.

Garbage Collection
==================
Garbage collection in Java is the process by which Java programs perform
automatic memory management.

Eventually, some objects will no longer be needed. The garbage collector
finds these unused objects and deletes them to free up memory.

The main objective of Garbage Collector is to free heap memory by
destroying unreachable objects.

The garbage collector always runs in the background.

It is an automatic process.

Student s = new Student("Yash");
s = new Student("Rahul");

new Student("Yash") is now dereferenced. It will automatically be
collected by Garbage Collector
-----X-----X-----X-----X-----
```