```
Relationships in Java
=====================
There are 2 types of relationships in Java:
1. Association
2. Inheritance

Association
===========
There are 2 types of associations:
1. Composition
2. Aggregation

Composition (part-of)
---------------------
Composition is when one class owns other classes and other classes can
not meaningfully exist, when the owner is destroyed.

ex: engine is a part-of Car. If the car is destroyed, engine will also be
destroyed.

Aggregation (has-a)
-------------------
In Aggregation, including objects can exist without being part of the
main object.

For ex: Student HAS-A Certificate
For ex: Student HAS-A Address

Anonymous Object
================
An Object that does not have a name (reference variable) is known as
anonymous object.

Ex: new Certificate("OCJA", 35000);

If we want to use an object name repeatedly we will assign a name to an
object.

Certificate cert1 = new Certificate("OCJP", 25000);

If we want to use an object only once, we can use anonymous object.
-----X-----X-----X-----
Inheritance (is-a)
==================
Inheritance is a mechanism in Java where a class object acquires all the
properties and behaviors of its parent class object.

Ex: Employee IS-A Person
Ex: Student IS-A Person
Ex: Maruti IS-A Car
Ex: WagonR IS-A Maruti

Employee is the child class.
A child class is also known as derived class, sub class
```

```
Person is the Parent class
A parent class is also known as base class, super class

Example:
Parent class -> Shape
Child class -> Rectangle extends Shape
Child class -> Triangle extends Shape
Grand Child Class -> Cuboid extends Reactangle

Types of Inheritance
--------------------
There are 3 types of inheritance:
1. Single Inheritance
2. Multilevel Inheritance
3. Hierarchical Inheritance

Refer figure: 01_inh_types.png

Single Inheritance
------------------
class Figure {

}

class Rectangle extends Figure {

}

Multilevel Inheritance
----------------------
class Cuboid extends Rectangle {

}

Hierarchichal Inheritance
-------------------------
class Circle extends Figure {

}

Inheritance Properties
----------------------
- The child class inherits variables & methods of parent class
- The child class do not inherit constructors of parent class
- The parent class cannot access any member of child classes

Constructor during Inheritance
==============================
- Constructors are not inherited by child classes.
- So the child classes have to define their own constructors for use.
- Whenever a constructor in child class is created it will automatically
call the default constructor of parent class whether we write it or not
- We can however call any other parent class constructor with super()
```

```
super keyoword
---------------
super is used to call parent class constructor, methods and variables

super(args-list) : constructor
super.method() : method
super.variable : variable

Runtime Polymorphism
====================
Upcasting
---------
A reference variable of a superclass can be assigned a reference to any
subclass object

For example:

class Figure {}

class Rectangle extends Figure {}

Figure f = new Figure();
Rectangle r = new Rectangle();
f = r;

This is known as Upcasting

Downcasting
-----------
A sub class cannot be assigned a super class reference directly.
However it can be done with downcasting.

Figure f = new Figure();
Rectangle r = new Rectangle();
r = (Rectangle)f;

This is known as Downcasting

Method Overriding
------------------
When a method in a subclass has the same name and same number/types of
parameter as a method in its super-class, then the method in the subclass
is said to override the method in the super-class.

Parent class reference can call the overriden method if it is storing
child class object.

Two necessary conditions of method overriding:
1. The method in child class should have the same signature as that in
parent class
2. The access modifier in the child class should have a visibity equal to
or greater than that of parent class method
```

```
@Override
---------
- The Override annotation monitors the above two necessary conditions for
method overriding.
- If either the signature of child class method does not match or access
is reduced, @Override shows an error

Polymorphism
============
Polymorphism means same name different functions

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
      - Static Polymorphism
      - Eager binding

Run Time Polymorphism
-------------------------
- If a parent class reference stores the address of a child class object,
then it can call the overriden methods in child class
- In Java, Run Time Polymorphism is achieved through method overriding.
- Here the call to the appropriate method is resolved at run time
- It is also known as
      - Dynamic binding
      - Late binding
      - Dynamic Polymorphism
      - Lazy binding

final keyword
=============
final can be used with:
1. Class - to stop inheritance
2. Method - to stop overriding
3. Variable - to stop re-assignment
-----X-----X-----X-----X-----
Abstraction
===========
Hiding the implementation is known as abstraction

There are 2 ways to achieve abstraction:

1. Abstract class
2. Interfaces
```

```
2 terms that we should know:

1. Concrete
      - A method with a definition is a concrete method
      - A class in which all methods are concrete is a concrete class
2. Abstract
      - A method without a definition is an abstract method
      - A class in which even 1 method is abstract is an abstract class

=> Concrete means non-abstract and abstract means non-concrete

Abstract Class
==============
A class that wants its child class to
- Reuse its functionality
- at the same time compulsorily define some methods is an Abstract class

Features
========
- An abstract class cannot be instantiated
- An abstract class can contain both abstract and concrete methods or one
of them or none of them
- An abstract class can have everything that a normal class has
      - Instance variables
      - Static variables
      - methods
      - constructors
      - etc
- The child MUST provide the implementation of all abstract methods in
parent class
- If the child class chooses not to define parent class abstract method,
then that child class too should be declared abstract

When to use Abstract classes
----------------------------
- When we want to reuse the properties and behavior of a parent class.
- When the parent class do not want to define a method on its own
- When we want to implement RTP
-----X-----X-----X-----X-----X-----
```