

Java Fundamentals and Object Oriented Programming

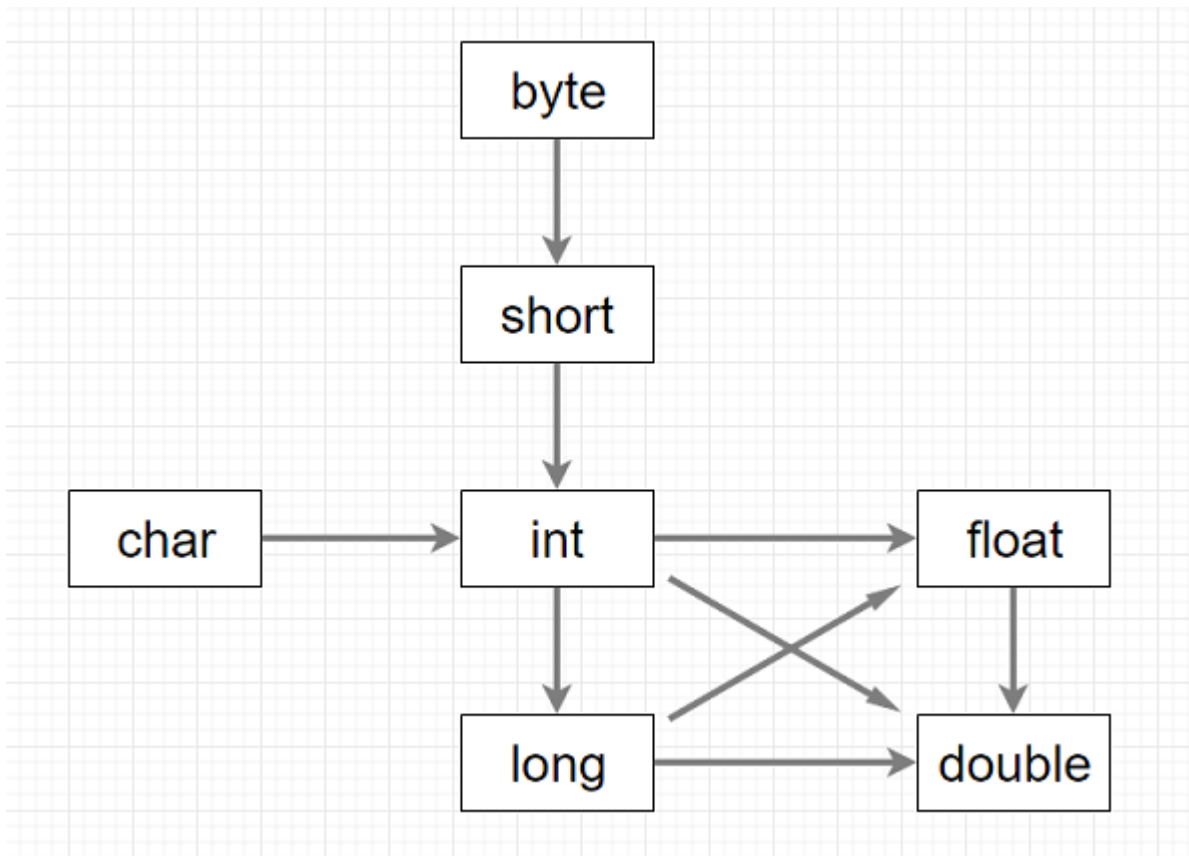
Object-Oriented Programming is a methodology or paradigm to design a program using classes and objects. It simplifies the software development and maintenance. Main Concepts - Inheritance, Polymorphism, Abstraction, Encapsulation.

Data Types in Java

| Data Type | Default Value | Default size |
|-----------|---------------|--------------|
| boolean | false | 1 bit |
| char | '\u0000' | 2 byte |
| byte | 0 | 1 byte |
| short | 0 | 2 byte |
| int | 0 | 4 byte |
| long | 0L | 8 byte |
| float | 0.0f | 4 byte |
| double | 0.0d | 8 byte |

NOTE : UTF-8 is the most popular unicode character encoding with 90% websites using it.

Data Type Promotion in Java :



Wrapper Classes in Java

| Primitive Type | Wrapper class |
|----------------|---------------|
| boolean | Boolean |
| char | Character |
| byte | Byte |
| short | Short |
| int | Integer |
| long | Long |
| float | Float |
| double | Double |

Operators in Java

| Operator Type | Category | Precedence |
|---------------|----------|------------|
|---------------|----------|------------|

| | | |
|------------|----------------------|---|
| Unary | postfix | <i>expr</i> ++ <i>expr</i> -- |
| | prefix | ++ <i>expr</i> -- <i>expr</i> + <i>expr</i> - <i>expr</i> ~ ! |
| Arithmetic | multiplicative | * / % |
| | additive | + - |
| Shift | shift | << >> >>> |
| Relational | comparison | < > <= >= instanceof |
| | equality | == != |
| Bitwise | bitwise AND | & |
| | bitwise exclusive OR | ^ |
| | bitwise inclusive OR | |
| Logical | logical AND | && |
| | logical OR | |
| Ternary | ternary | ? : |
| Assignment | assignment | = += -= *= /= %= &= ^= = <<= >>= >>>= |

Java Naming Conventions :

| Name | Convention |
|----------------|---|
| class name | should start with uppercase letter and be a noun e.g. String, Color, Button, System, Thread etc. |
| interface name | should start with uppercase letter and be an adjective e.g. Runnable, Remote, ActionListener etc. |
| method name | should start with lowercase letter and be a verb e.g. actionPerformed(), main(), print(), println() etc. |
| variable name | should start with lowercase letter e.g. firstName, orderNumber etc. |
| package name | should be in lowercase letter e.g. java, lang, sql, util etc. |

| | |
|----------------|---|
| constants name | should be in uppercase letter. e.g. RED, YELLOW, MAX_PRIORITY etc. |
|----------------|---|

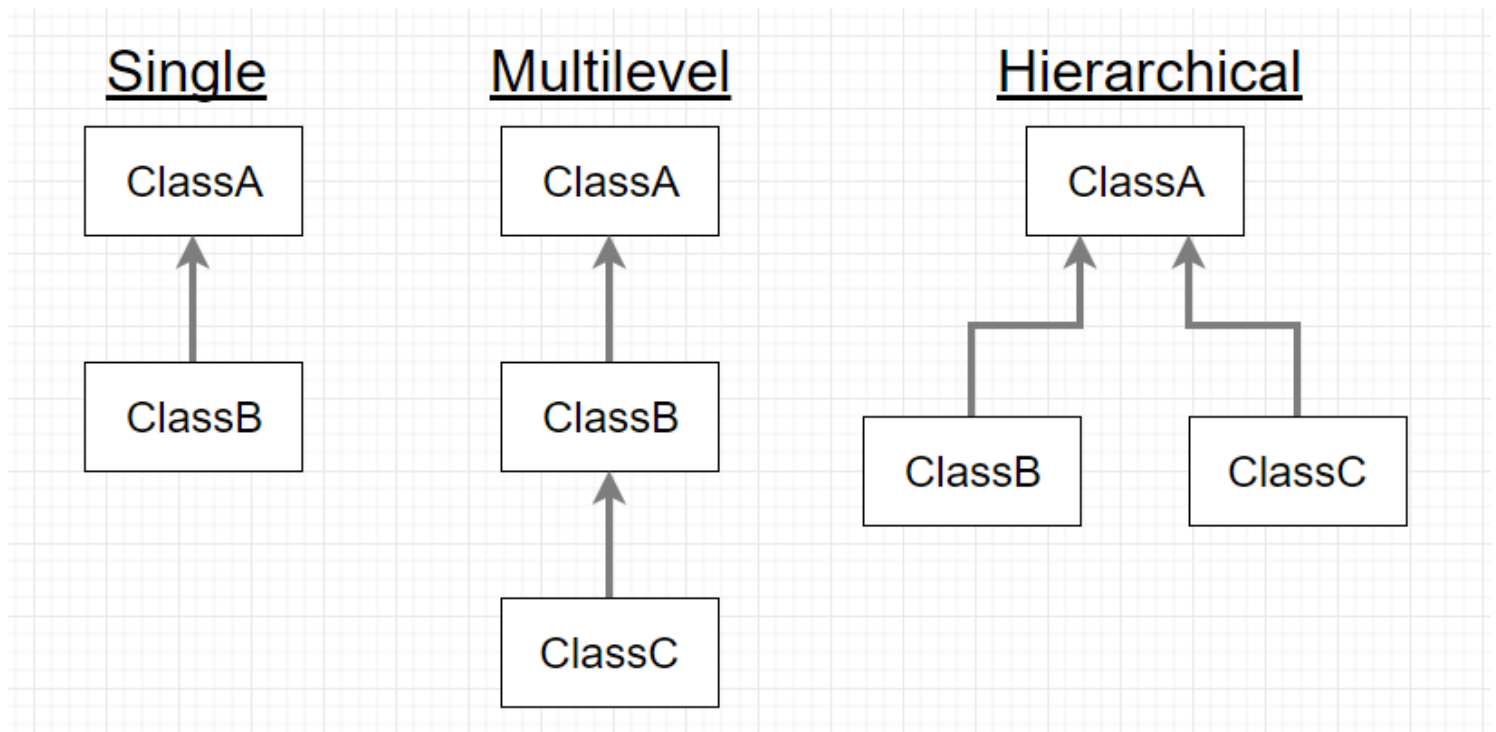
Object vs Class

| Object | Class |
|---|---|
| Object is an instance of a class. | Class is a blueprint or template from which objects are created. |
| Object is a real world entity such as pen, laptop, mobile, bed, keyboard, mouse, chair etc. | Class is a group of similar objects . |
| Object is a physical entity. | Class is a logical entity. |
| Object is created through new keyword mainly e.g. Student s1=new Student(); | Class is declared using class keyword e.g. class Student{} |
| Object is created many times as per requirement. | Class is declared once . |
| Object allocates memory when it is created. | Class doesn't allocated memory when it is created. |
| There are many ways to create object like new keyword, newInstance() method, clone() method, factory method & deserialization. | There is only one way to define class in java using class keyword. |

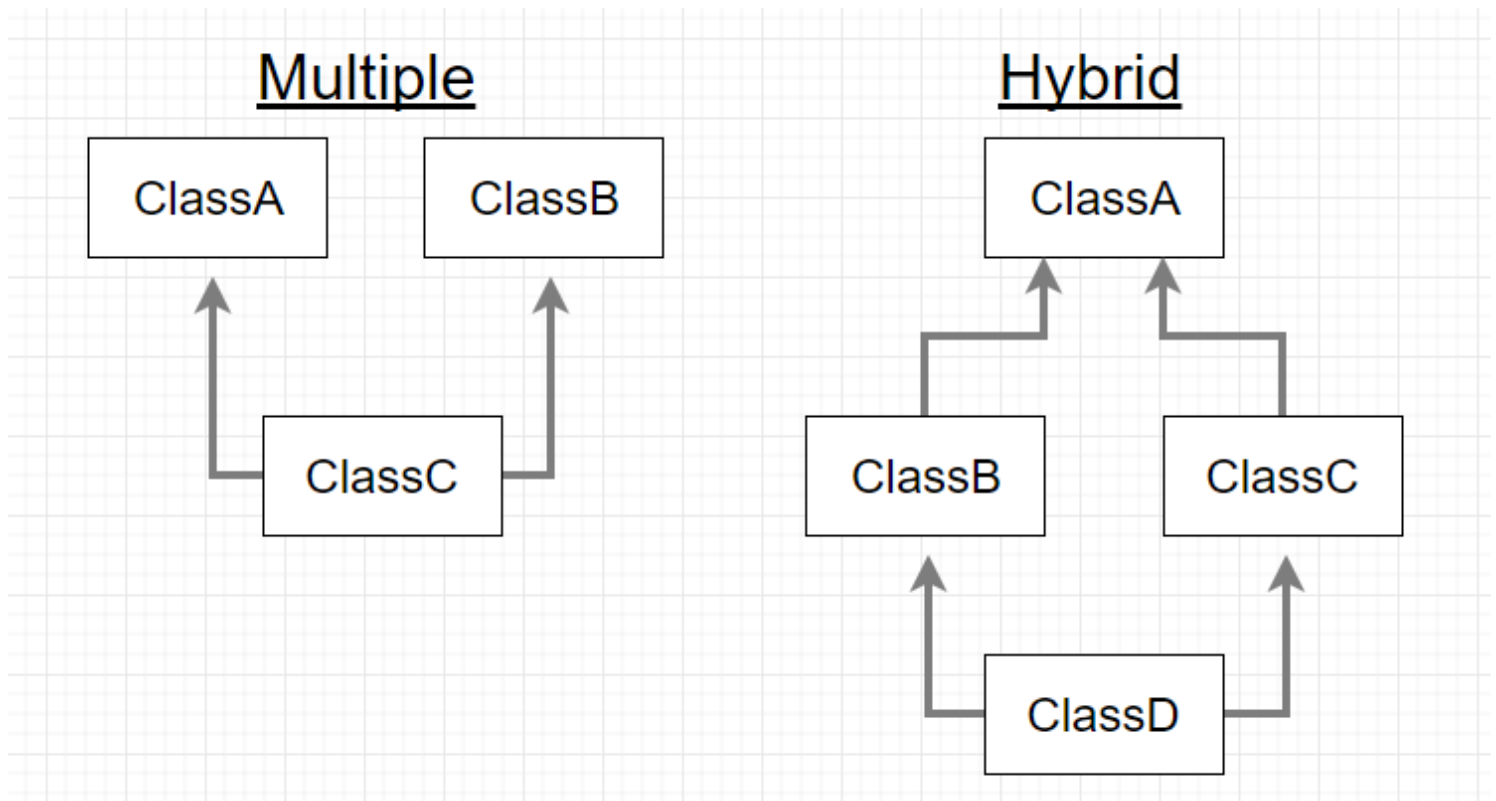
Constructors vs Methods

| Java Constructor | Java Method |
|--|---|
| Constructor is used to initialize the state of an object. | Method is used to expose behaviour of an object. |
| Constructor must not have return type. | Method must have return type. |
| Constructor is invoked implicitly. | Method is invoked explicitly. |
| Compiler provides a default constructor if you don't have any constructor. | Method is not provided by compiler in any case. |
| Constructor name must be same as the class name. | Method name may or may not be same as class name. |

Types of Inheritance (Supported through Class)



Types of Inheritance (Supported through Interface only)



Association vs Aggregation vs Composition

Association

Aggregation

(WEAK ASSOCIATION)

Composition

(STRONG ASSOCIATION)

Aggregation vs Composition

| Aggregation | Composition |
|--|--|
| Aggregation is a weak Association. | Composition is a strong Association. |
| Class can exist independently without owner. | Class can not meaningfully exist without owner. |
| Have their own Life Time. | Life Time depends on the Owner. |
| A uses B. | A owns B. |
| Child is not owned by 1 owner. | Child can have only 1 owner. |
| Has-A relationship. A has B. | Part-Of relationship. B is part of A. |
| Denoted by a empty diamond in UML. | Denoted by a filled diamond in UML. |
| We do not use "final" keyword for Aggregation. | "final" keyword is used to represent Composition. |
| Examples: <ul style="list-style-type: none">- Car has a Driver.- A Human uses Clothes.- A Company is an aggregation of People.- A Text Editor uses a File.- Mobile has a SIM Card. | Examples: <ul style="list-style-type: none">- Engine is a part of Car.- A Human owns the Heart.- A Company is a composition of Accounts.- A Text Editor owns a Buffer.- IMEI Number is a part of a Mobile. |

NOTE : "final" keyword is used in Composition to make sure child variable is initialized.

Polymorphism - Method Overloading vs Method Overriding

| Method Overloading | Method Overriding |
|---|--|
| Method overloading is used <i>to increase the readability</i> of the program. | Method overriding is used <i>to provide the specific implementation</i> of the method that is already provided by its super class. |
| Method overloading is performed <i>within class</i> . | Method overriding occurs <i>in two classes</i> that have IS-A (inheritance) relationship. |
| In case of method overloading, <i>parameter must be different</i> . | In case of method overriding, <i>parameter must be same</i> . |
| Method overloading is the example of <i>compile time polymorphism</i> . | Method overriding is the example of <i>run time polymorphism</i> . |
| In java, method overloading can't be done by changing only the return type of method. <i>Return type can be same/different</i> in overloading, but you must change the parameter. | <i>Return type must be same or covariant (changing return type to subclass type)</i> in method overriding. |

Abstract Class vs Interface

| Abstract class | Interface |
|--|---|
| Abstract class can have abstract and non-abstract methods. | Interface can have only abstract methods. Since Java8, it can have default & static methods also. |
| Abstract class doesn't support multiple inheritance . | Interface supports multiple inheritance . |
| Abstract class can have final, non-final, static and non-static variables . | Interface has only static and final variables . |
| Abstract class can provide the implementation of interface . | Interface can't provide the implementation of abstract class . |
| The abstract keyword is used to declare abstract class. | The interface keyword is used to declare interface. |
| Example: <pre>public abstract class Shape{ public abstract void draw();}</pre> | Example: <pre>public interface Drawable{ void draw();}</pre> |

Java Access Modifiers

| Access Modifier | within class | within package | outside package by subclass only | outside package |
|-----------------|--------------|----------------|----------------------------------|-----------------|
| Private | Y | N | N | N |
| Default | Y | Y | N | N |
| Protected | Y | Y | Y | N |
| Public | Y | Y | Y | Y |

Abstraction vs Encapsulation

| Abstraction | Encapsulation |
|---|---|
| Abstraction is a process of hiding the implementation details and showing only functionality to the user. | Encapsulation is a process of wrapping code and data together into a single unit |
| Abstraction lets you focus on what the object does instead of how it does it. | Encapsulation provides you the control over the data and keeping it safe from outside misuse. |
| Abstraction solves the problem in the Design Level. | Encapsulation solves the problem in the Implementation Level. |
| Abstraction is implemented by using Interfaces and Abstract Classes. | Encapsulation is implemented by using Access Modifiers (private, default, protected, public) |
| Abstraction means hiding implementation complexities by using interfaces and abstract class. | Encapsulation means hiding data by using setters and getters. |

Methods of Object Class

The Object class is the parent class of all the classes in java by default.

| Method | Description |
|-------------------------------|--|
| public final Class getClass() | returns the Class class object of this object. The Class class can further be used to get the metadata of this |

| | |
|---|---|
| | class. |
| public int hashCode() | returns the hashcode number for this object. |
| public boolean equals(Object obj) | compares the given object to this object. |
| protected Object clone() throws CloneNotSupportedException | creates and returns the exact copy (clone) of this object. |
| public String toString() | returns the string representation of this object. |
| public final void notify() | wakes up single thread, waiting on this object's monitor. |
| public final void notifyAll() | wakes up all the threads, waiting on this object's monitor. |
| public final void wait(long timeout)throws InterruptedException | causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method). |
| public final void wait(long timeout,int nanos)throws InterruptedException | causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method). |
| public final void wait()throws InterruptedException | causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method). |
| protected void finalize()throws Throwable | is invoked by the garbage collector before object is being garbage collected. |