



Java was created by
James Arthur Gosling - 1995

INTRODUCTION TO JAVA

Why Java?

- Open source, free
- Easy to implement
- Secure, robust & powerful
- Object oriented
- High-level language
- Platform-independent, Portable

Environment & Tools

- Processor speed 3.6 GHz +
- 8 GB + RAM
- JDK (Java Development Kit)
- IDE (Integrated Dev Tool)
(IntelliJ /Eclipse/ Net Beans)
- JVM (Java Virtual Machine)

Download IntelliJ IDEA – The Leading IDE

jetbrains.com/idea/download/#section=windows

IntelliJ IDEA

Download Pricing

Download IntelliJ IDEA

Windows macOS Linux

Ultimate
The Leading Java and Kotlin IDE

Download .exe

Free 30-day trial available

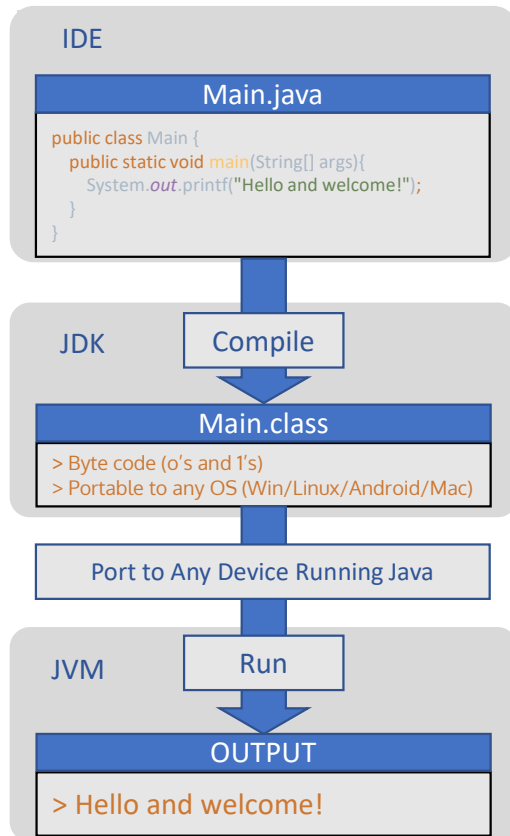
Community Edition
The IDE for pure Java and Kotlin development

Download .exe

Free, built on open source

Version: 2023.1.1
Build: 231.8770.65
28 April 2023
[Release notes](#)
[System requirements](#)

JVM (Java Virtual Machine)



JVM Running as Windows Process

Task Manager

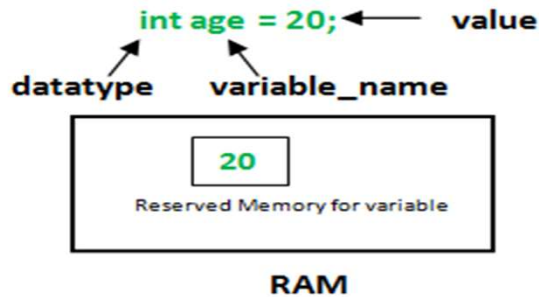
Type a name, publisher, or PID to s...

Processes

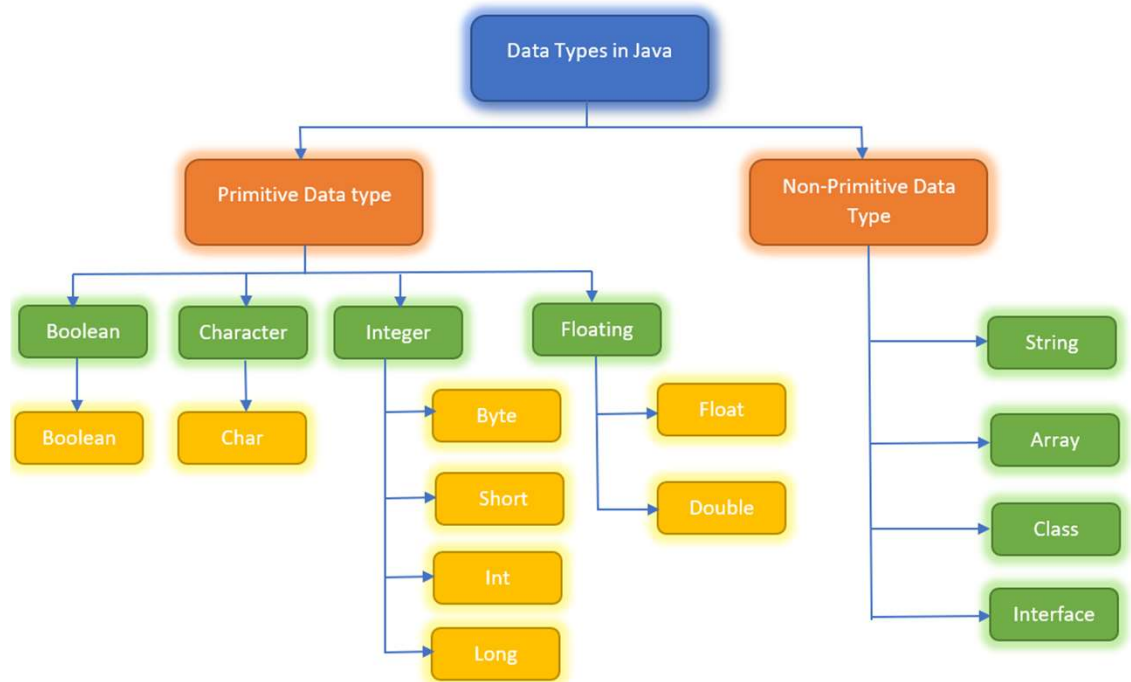
Run new task End task Efficiency mode ...

Name	Status	4% CPU	47% Memory	1% Disk	0% Network
Apps (7)					
> Google Chrome (12)		0%	603.3 MB	0.1 MB/s	0 Mbps
> IntelliJ IDEA (7)		0%	1,442.3 MB	0 MB/s	0 Mbps
Console Window Host		0%	0.3 MB	0 MB/s	0 Mbps
Console Window Host		0%	4.8 MB	0 MB/s	0 Mbps
Console Window Host		0%	0.1 MB	0 MB/s	0 Mbps
Filesystem events process...		0%	0.1 MB	0 MB/s	0 Mbps
IntelliJ IDEA		0%	1,347.9 MB	0 MB/s	0 Mbps
OpenJDK Platform binary		0%	84.9 MB	0 MB/s	0 Mbps
Windows PowerShell		0%	4.2 MB	0 MB/s	0 Mbps
> Microsoft PowerPoint (3)		0%	136.9 MB	0 MB/s	0 Mbps
> Paint		0%	35.6 MB	0 MB/s	0 Mbps
> Task Manager		0%	56.0 MB	0 MB/s	0 Mbps
> WhatsApp (2)		0%	99.0 MB	0 MB/s	0 Mbps
> Windows Explorer		0%	56.0 MB	0 MB/s	0 Mbps
Background processes (79)					

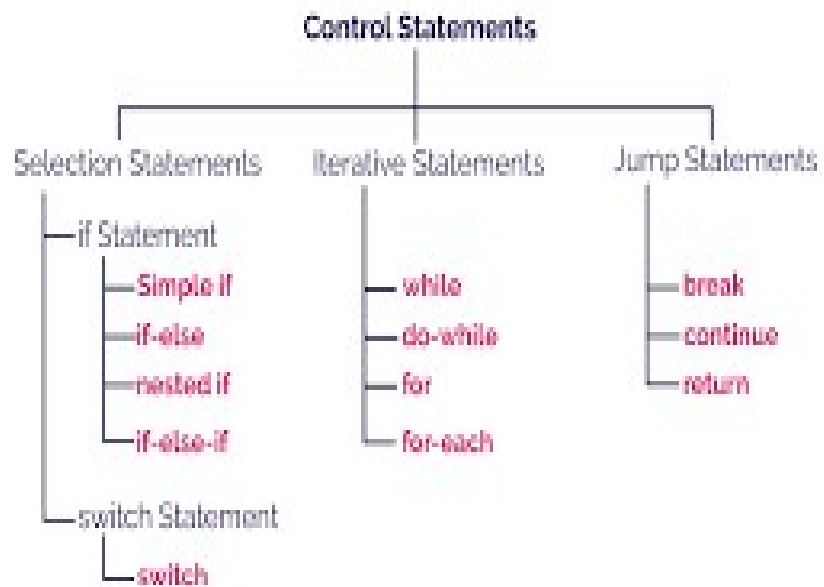
Variables, Data Types and Arrays



Single Dimensional Array In Java



Control Flow Statements and Methods



www.anodiam.com

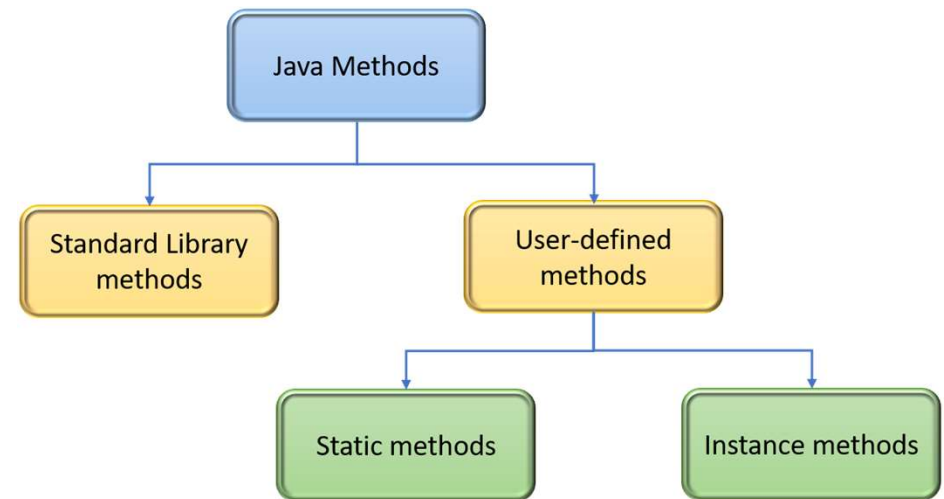


Diagram illustrating the syntax of a Java method signature and body:

```

public int getSum (int a, int b) {
    int sum;
    sum = a + b;
    return sum;
}
  
```

Labels in the diagram:

- modifier**: points to `public`
- return type**: points to `int`
- method name**: points to `getSum`
- list of parameters**: points to `(int a, int b)`
- method body**: points to the code block inside the curly braces.

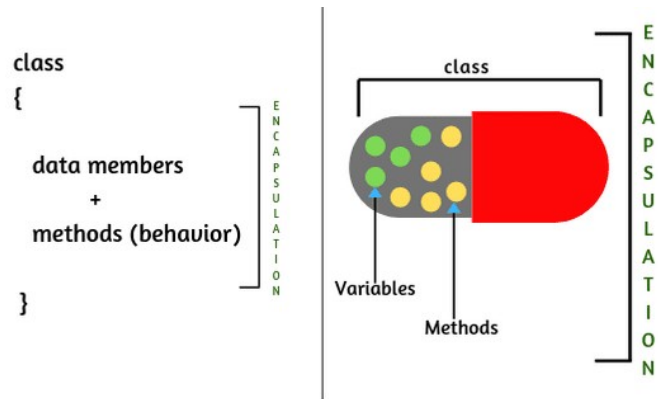
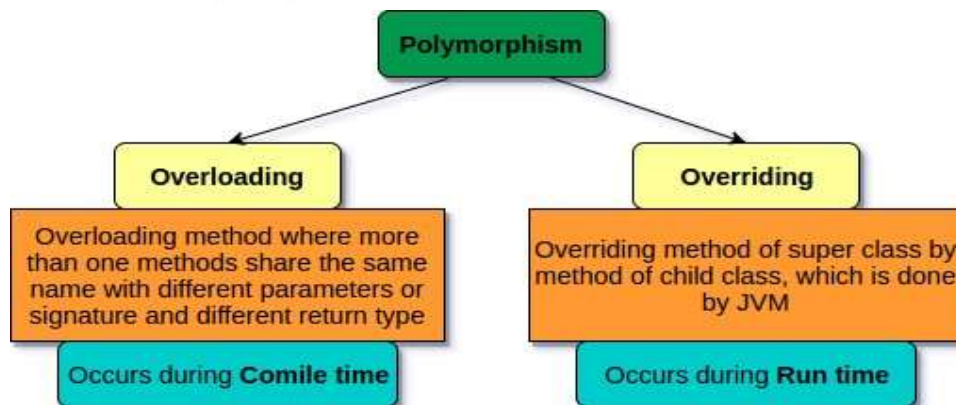
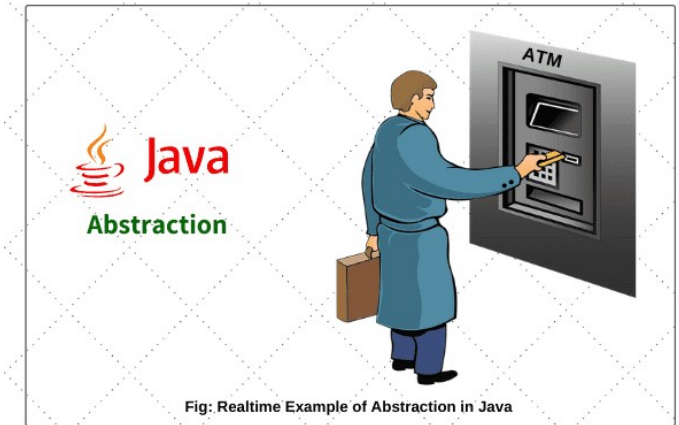
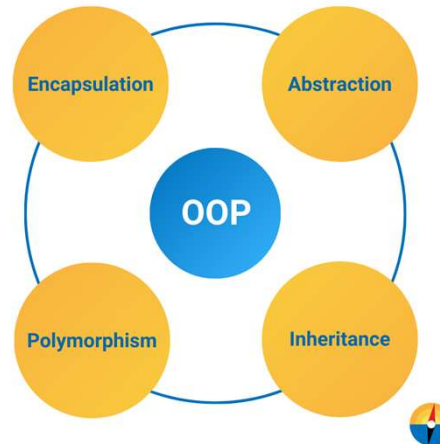


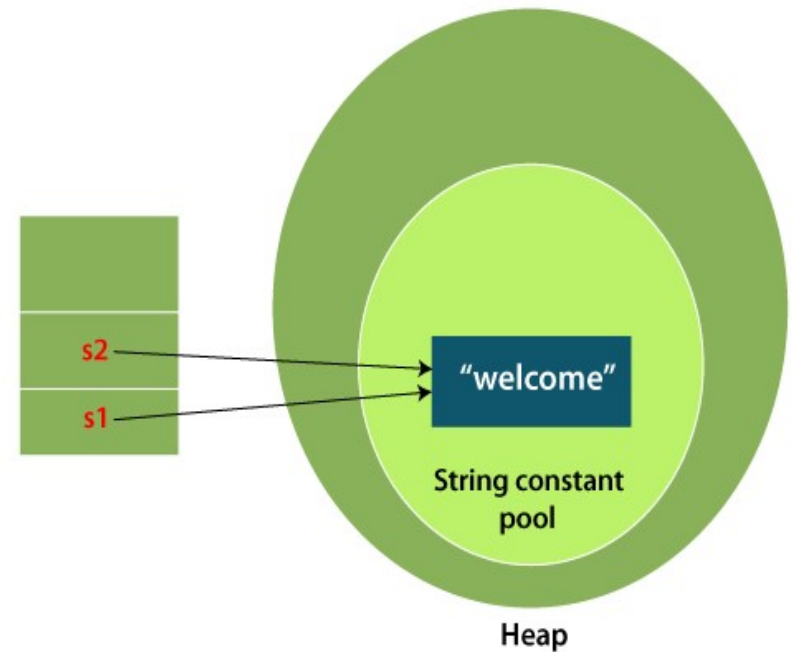
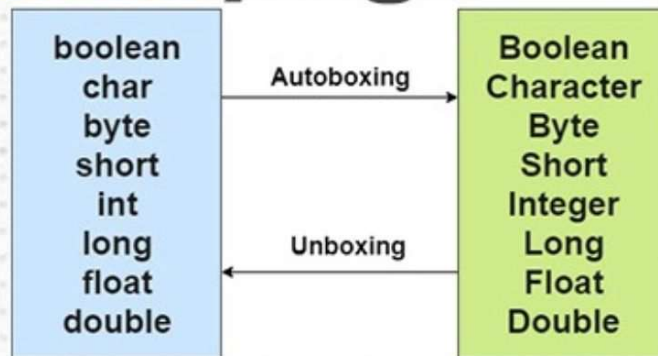
Fig: Encapsulation

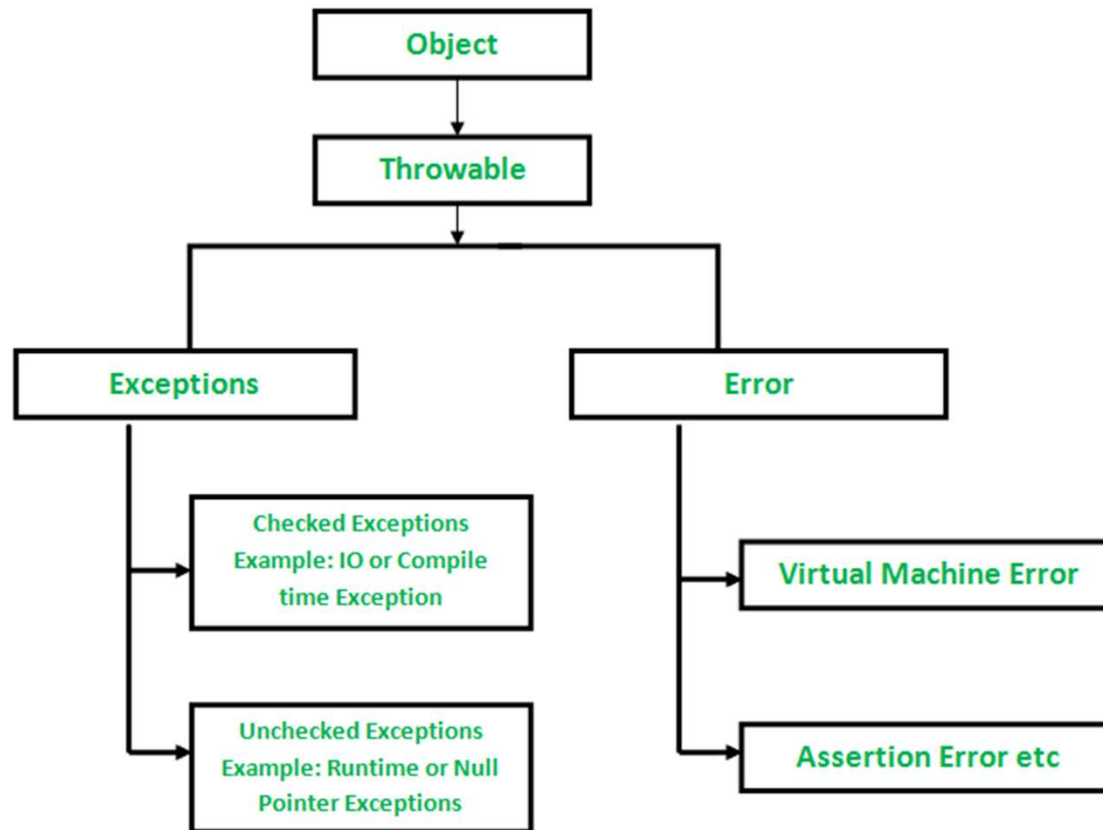


WRAPPER CLASSES

AUTOBOXING & UNBOXING

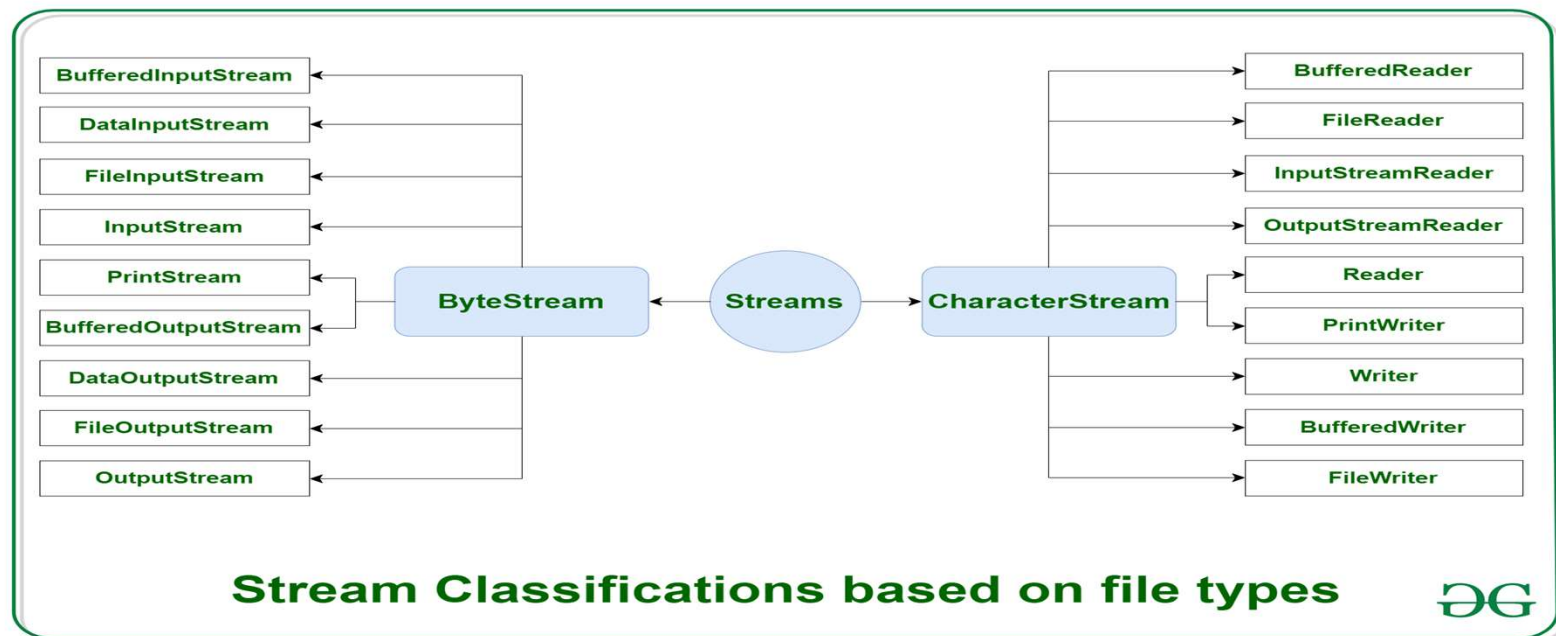
in **JAVA** programming

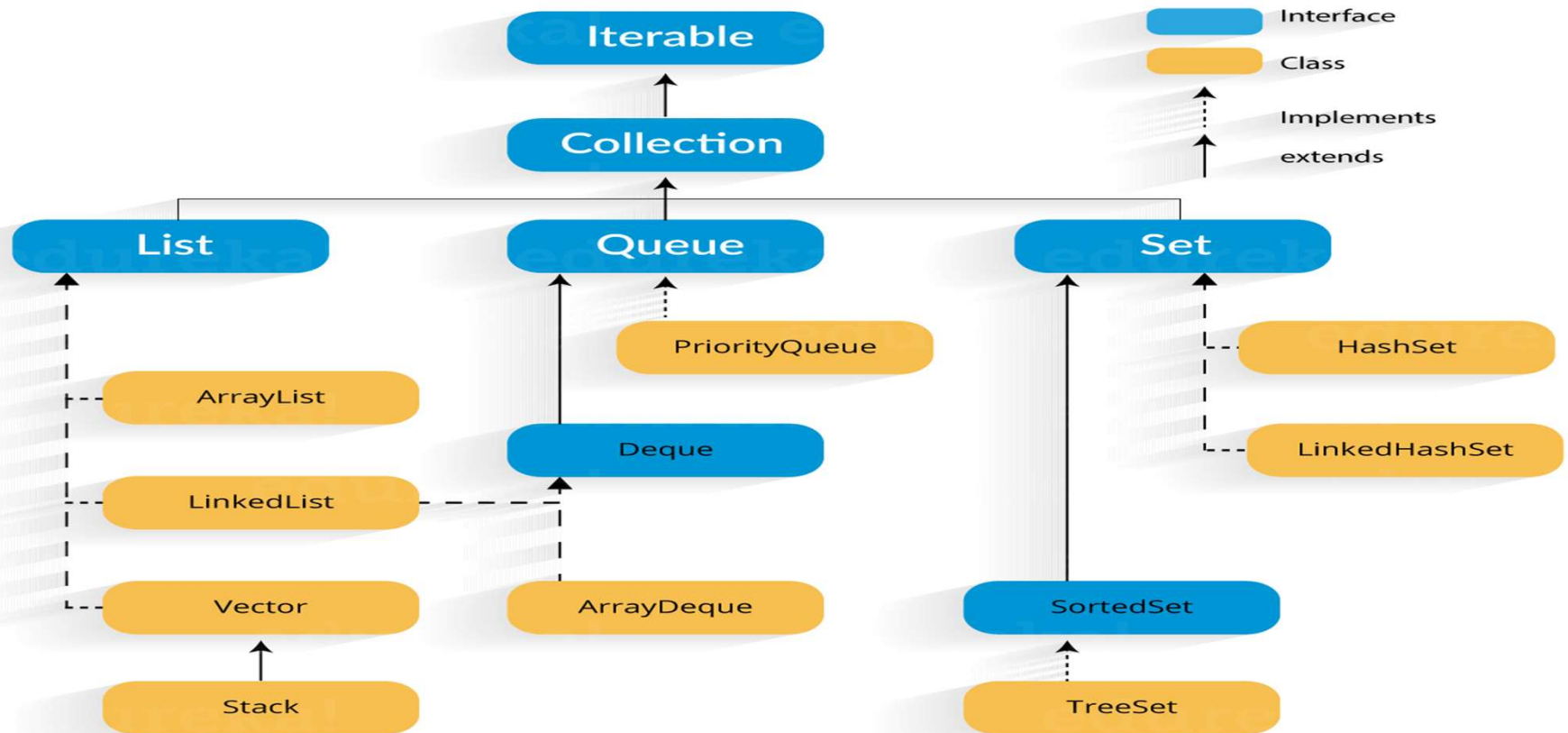




Date-time classes in Java	Legacy class	Modern class
Moment in UTC	<code>java.util. Date</code> <code>java.sql. Timestamp</code>	<code>java.time. Instant</code>
Moment with offset-from-UTC (hours-minutes-seconds)	(lacking)	<code>java.time. OffsetDateTime</code>
Moment with time zone ("Continent/Region")	<code>java.util. GregorianCalendar</code> <code>javax.xml.datatype. XMLGregorianCalendar</code>	<code>java.time. ZonedDateTime</code>
Date & Time-of-day (no offset, no zone) Not a moment	(lacking)	<code>java.time. LocalDateTime</code>

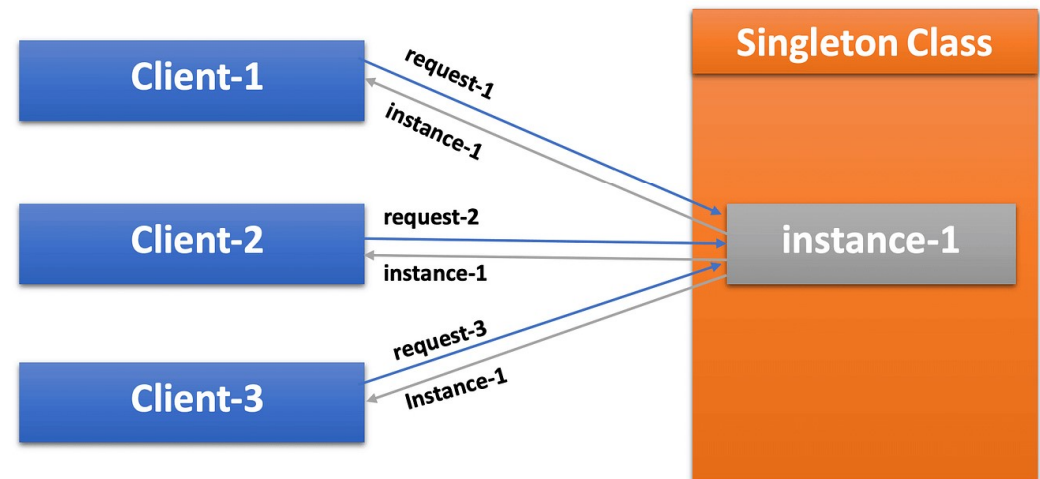
Java Input/Output





Singleton pattern is a design pattern which restricts a class to instantiate its multiple objects. It is nothing but a way of defining a class. Class is defined in such a way that only one instance of the class is created in the complete execution of a program or project. It is used where only a single instance of a class is required to control the action throughout the execution. A singleton class shouldn't have multiple instances in any case and at any cost. Singleton classes are used for logging, driver objects, caching and thread pool, database connections.

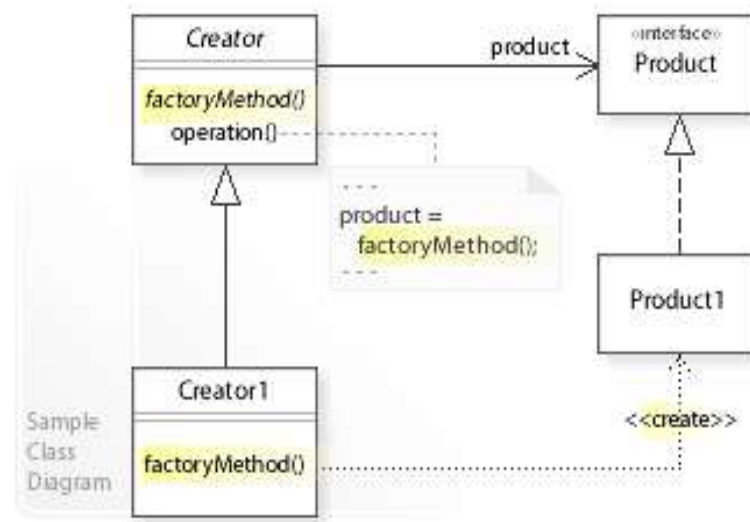
Singleton design pattern



Factory Pattern

Creating an object often requires complex processes not appropriate to include within a composing object. The object's creation may lead to a significant duplication of code, may require information not accessible to the composing object, may not provide a sufficient level of abstraction, or may otherwise not be part of the composing object's [concerns](#). The factory method design pattern handles these problems by defining a separate [method](#) for creating the objects, which [subclasses](#) can then override to specify the [derived type](#) of product that will be created.

The factory method pattern relies on inheritance, as object creation is delegated to subclasses that implement the factory method to create objects.



Observer Pattern

The observer design pattern is a behavioural pattern listed among the 23 well-known ["Gang of Four" design patterns](#) that address recurring design challenges in order to design flexible and reusable object-oriented software, yielding objects that are easier to implement, change, test and reuse.

In [software design](#) and [engineering](#), the **observer pattern** is a [software design pattern](#) in which an **object**, named the **subject**, maintains a list of its dependents, called **observers**, and notifies them automatically of any [state changes](#), usually by calling one of their [methods](#). It is often used for implementing distributed [event-handling](#) systems in [event-driven software](#). In such systems, the subject is usually named a "stream of events" or "stream source of events" while the observers are called "sinks of events." The stream nomenclature alludes to a physical setup in which the observers are physically separated and have no control over the emitted events from the subject/stream source. This pattern thus suits any process by which data arrives from some input that is not available to the [CPU](#) at [startup](#), but instead arrives seemingly at random ([HTTP requests](#), [GPIO](#) data, [user input](#) from [peripherals](#), [distributed databases](#) and [blockchains](#), etc.).

