Anodiam
Education that enlightens!

INTRODUCTION TO JAVA

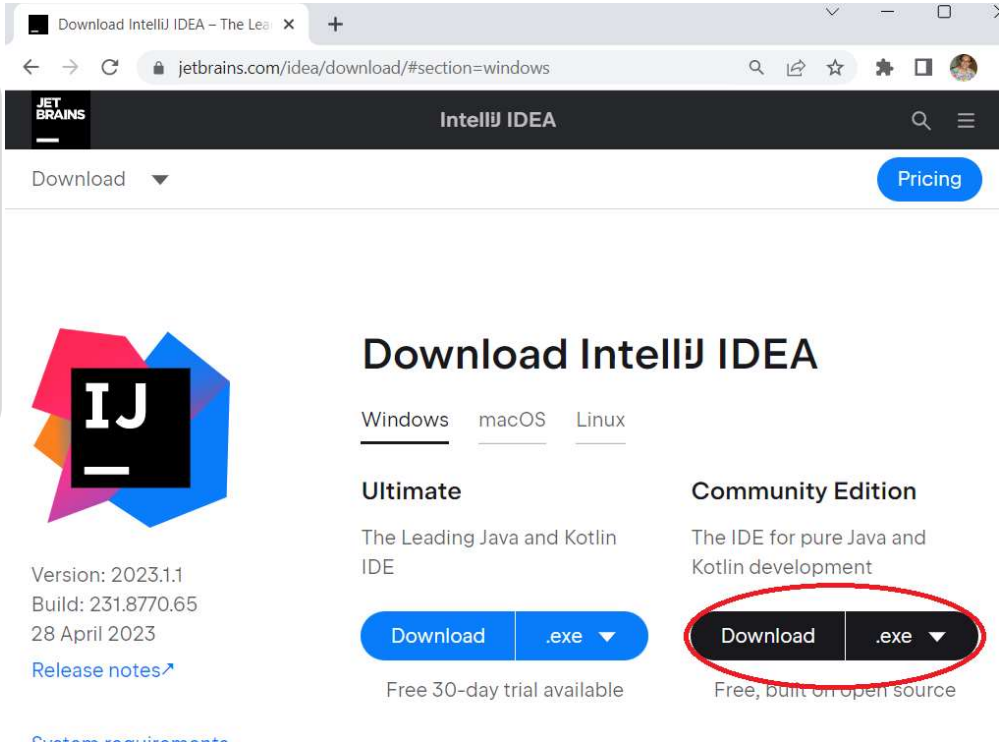
Java was created by
James Arthur Gosling - 1995

Why Java?

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

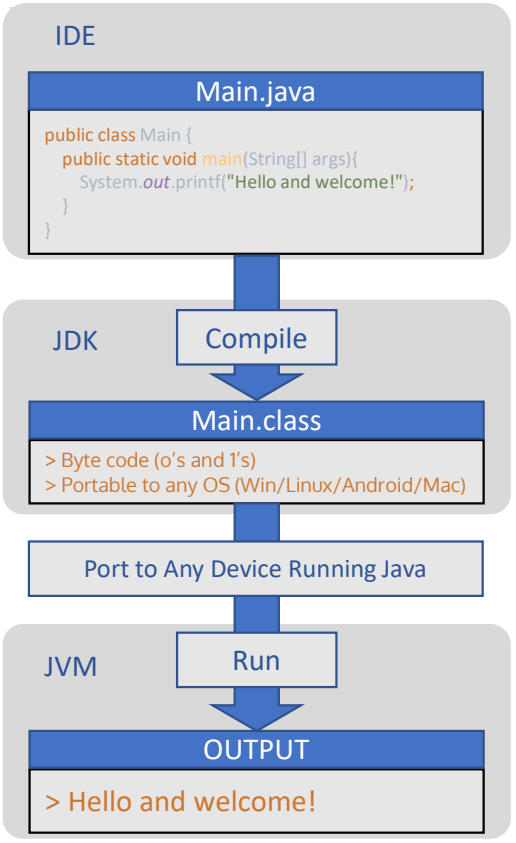
Environment & Tools

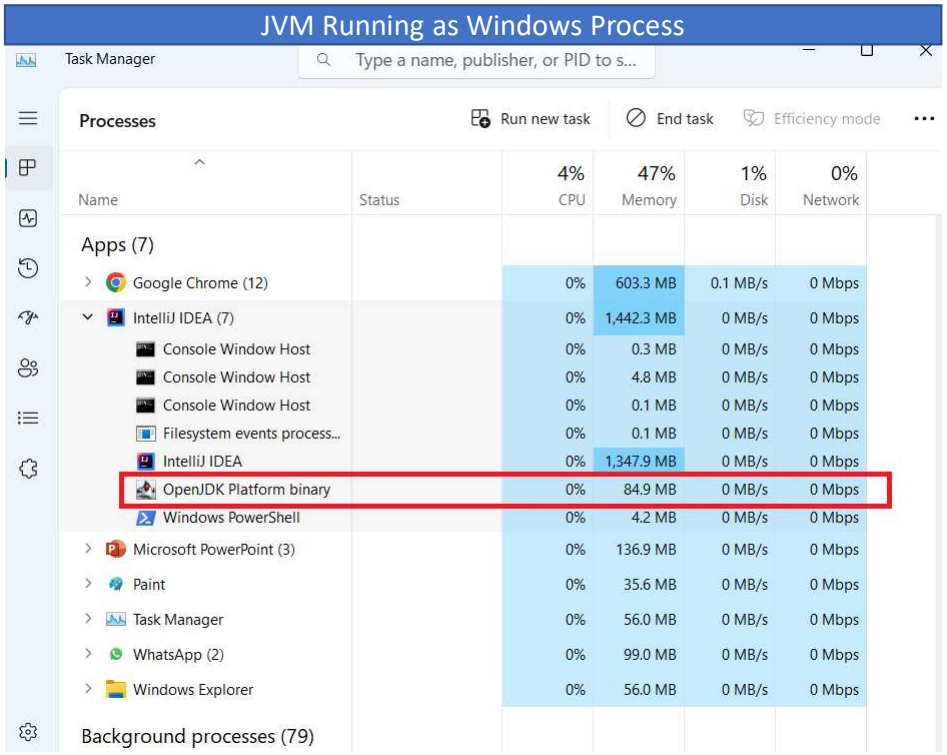
- Processor speed 3.6 GHz +
- 8 GB + RAM
- JDK (Java Development Kit)
- IDE (Integrated Dev Tool)
  (IntelliJ /Eclipse/ Net Beans)
- JVM (Java Virtual Machine)



Education that enlightens!

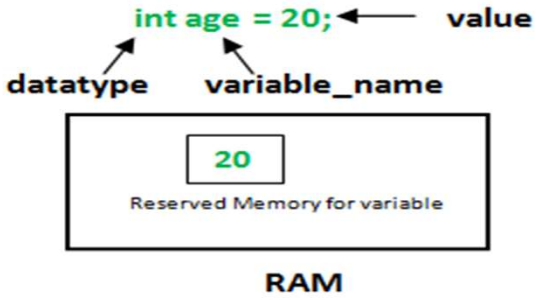
JVM (Java Virtual Machine)

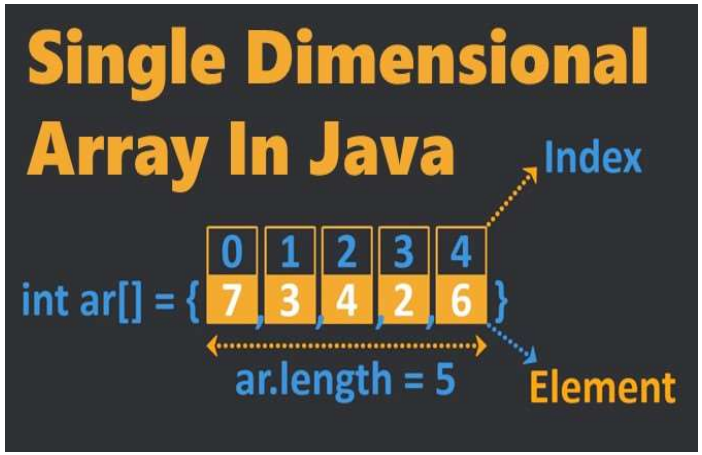


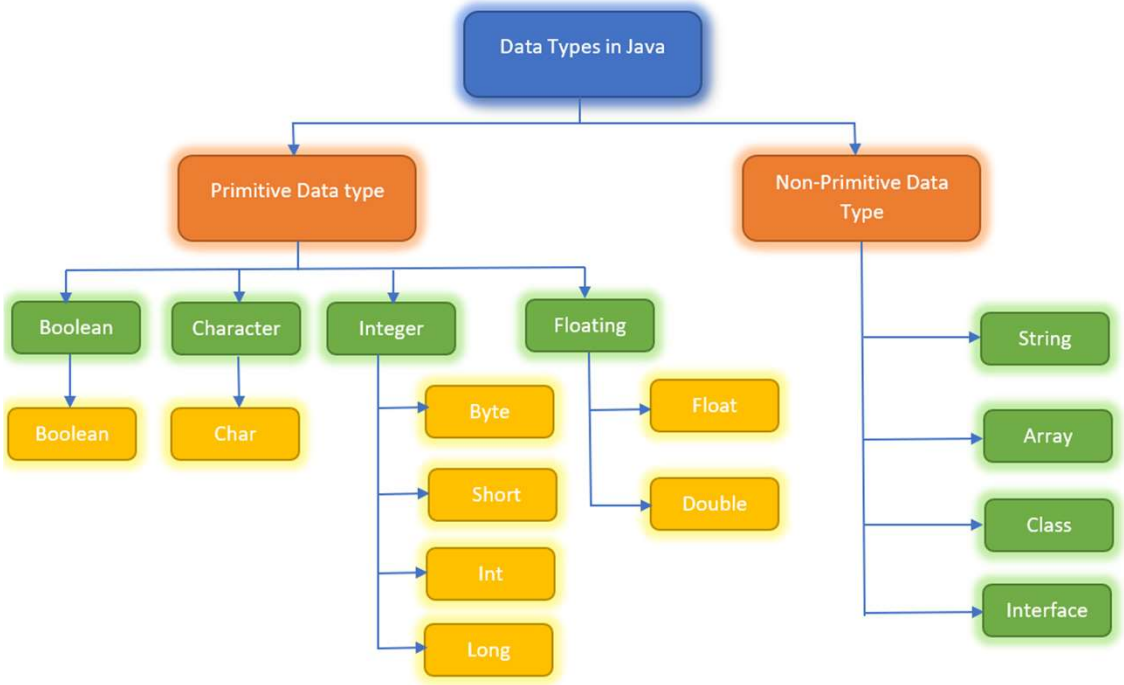


Education that enlightens!

Variables, Data Types and Arrays



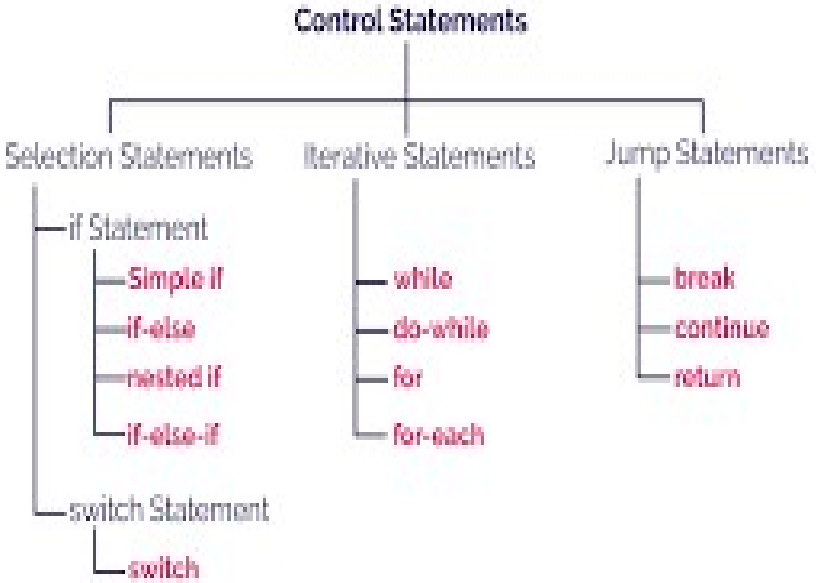


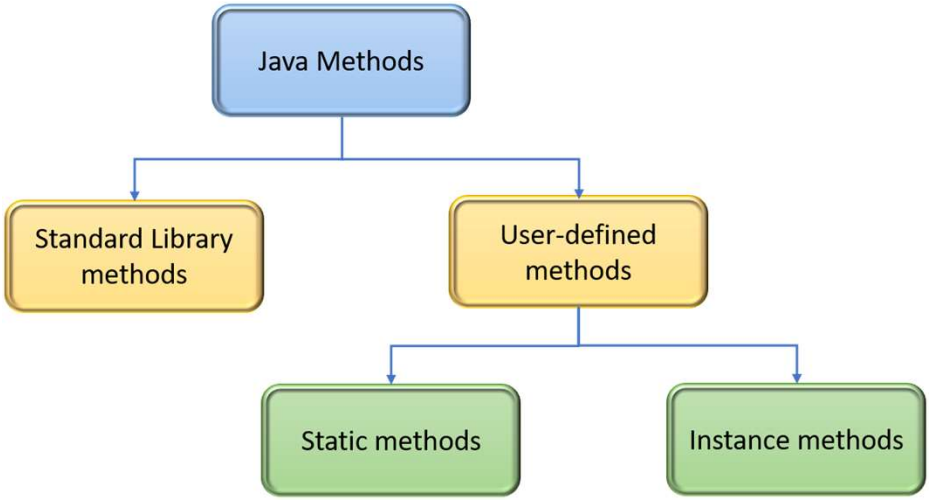


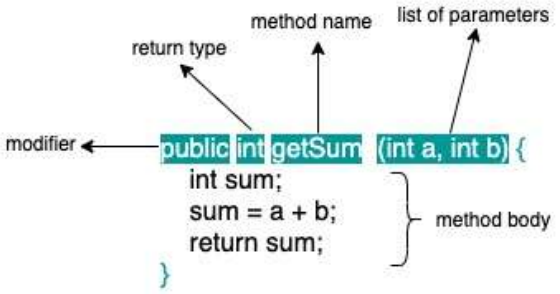
anodiam

Education that enlightens!

Control Flow Statements and Methods







Education that enlightens!

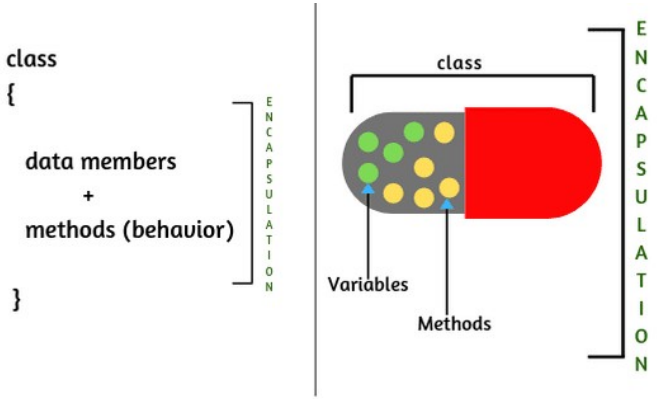
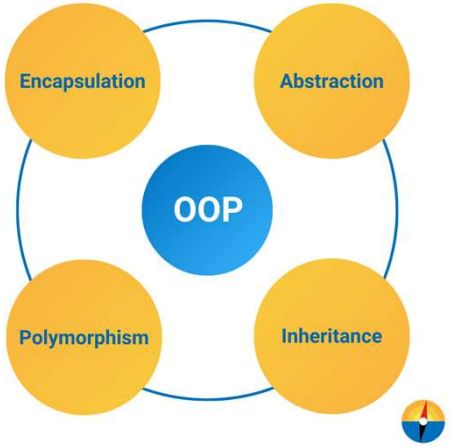


Fig: Encapsulation



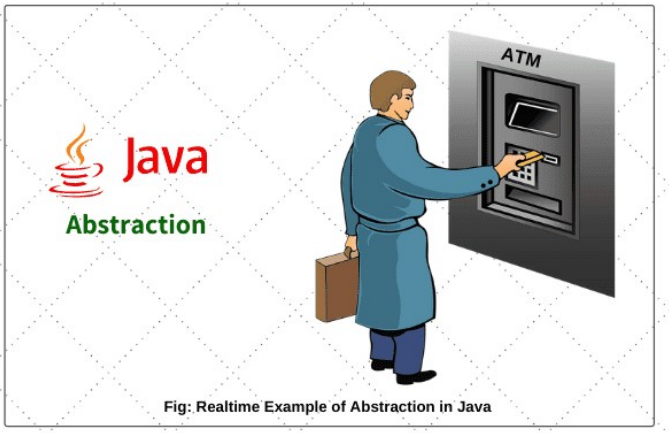
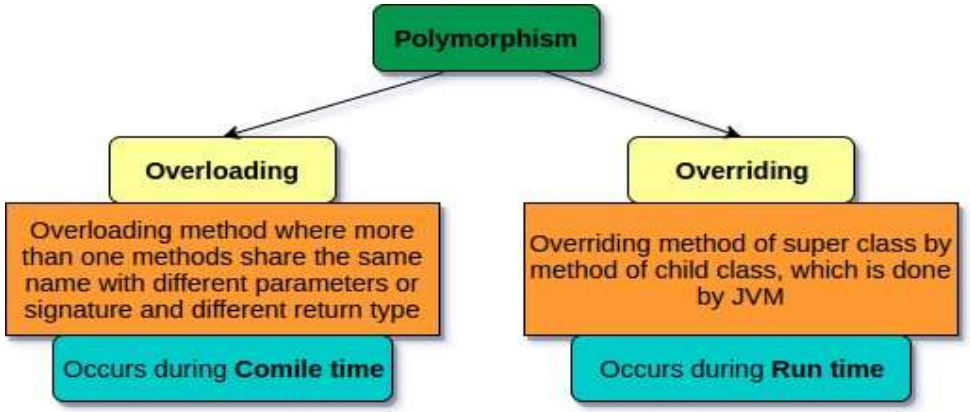
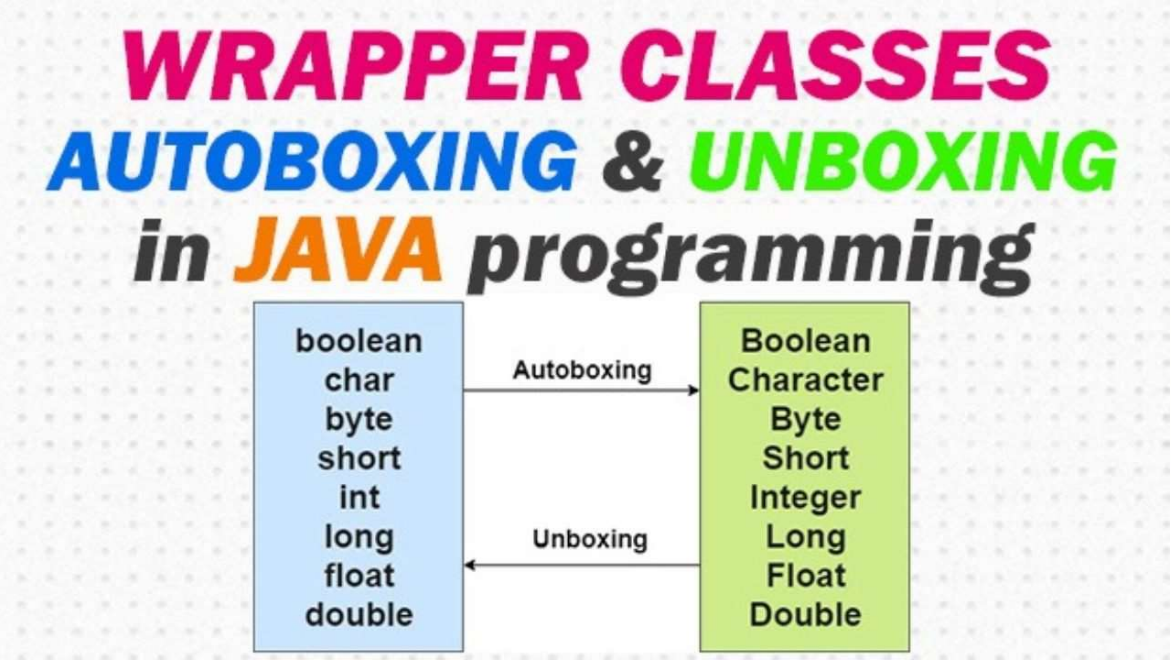
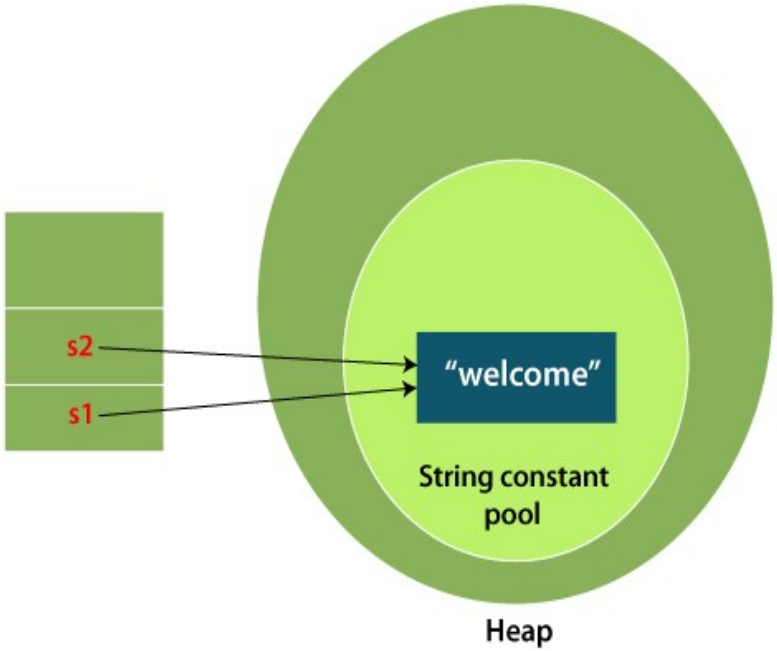


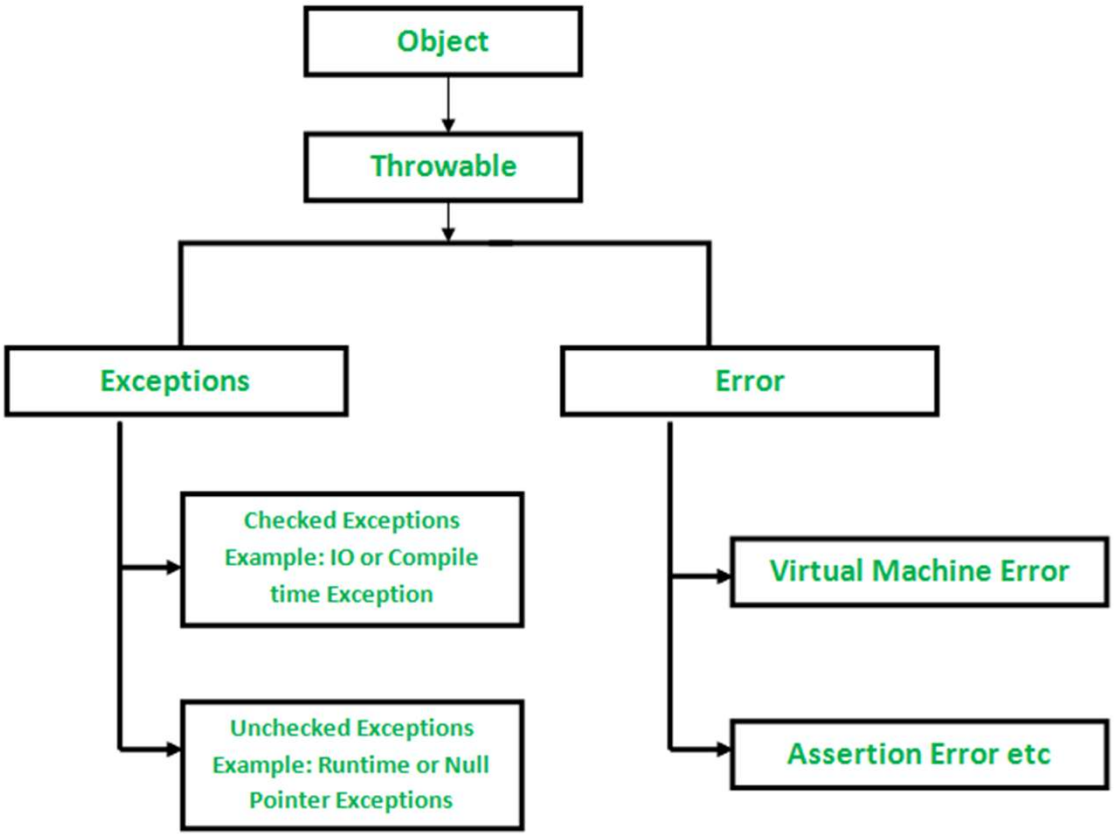
Fig: Realtime Example of Abstraction in Java



Education that enlightens!

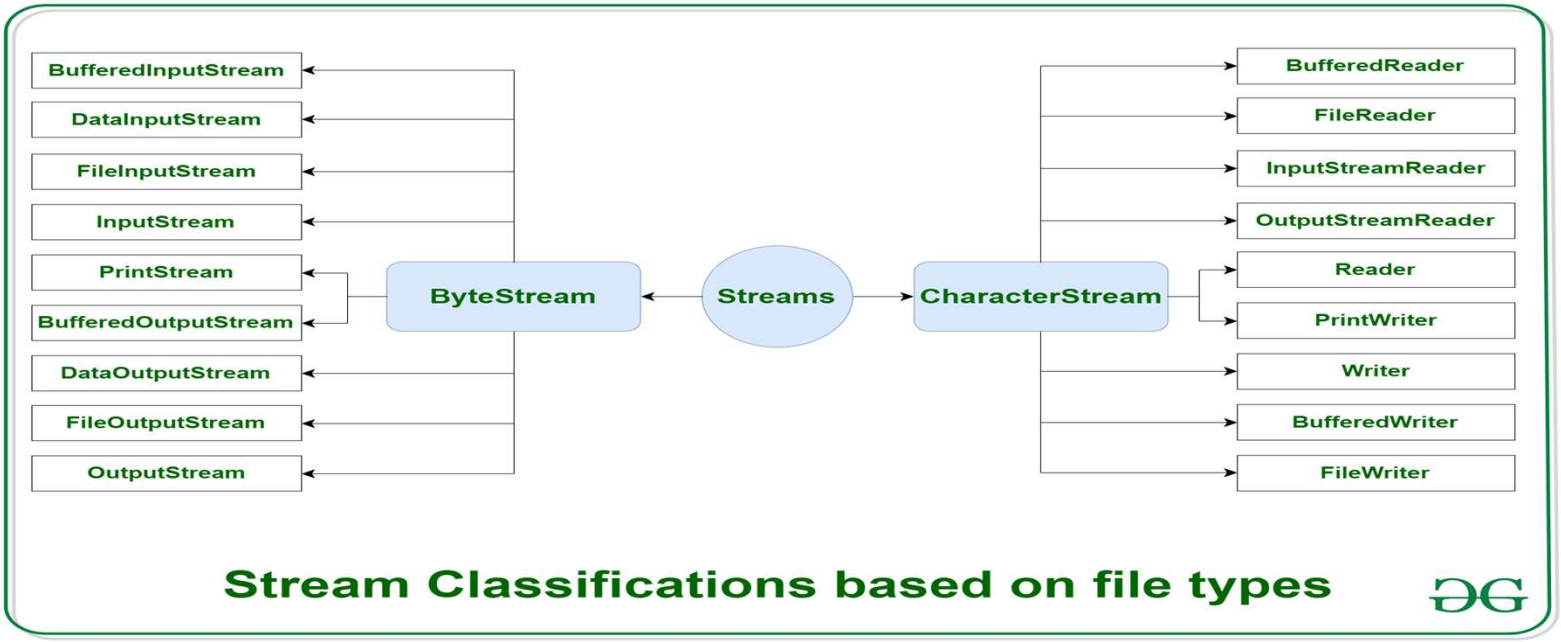


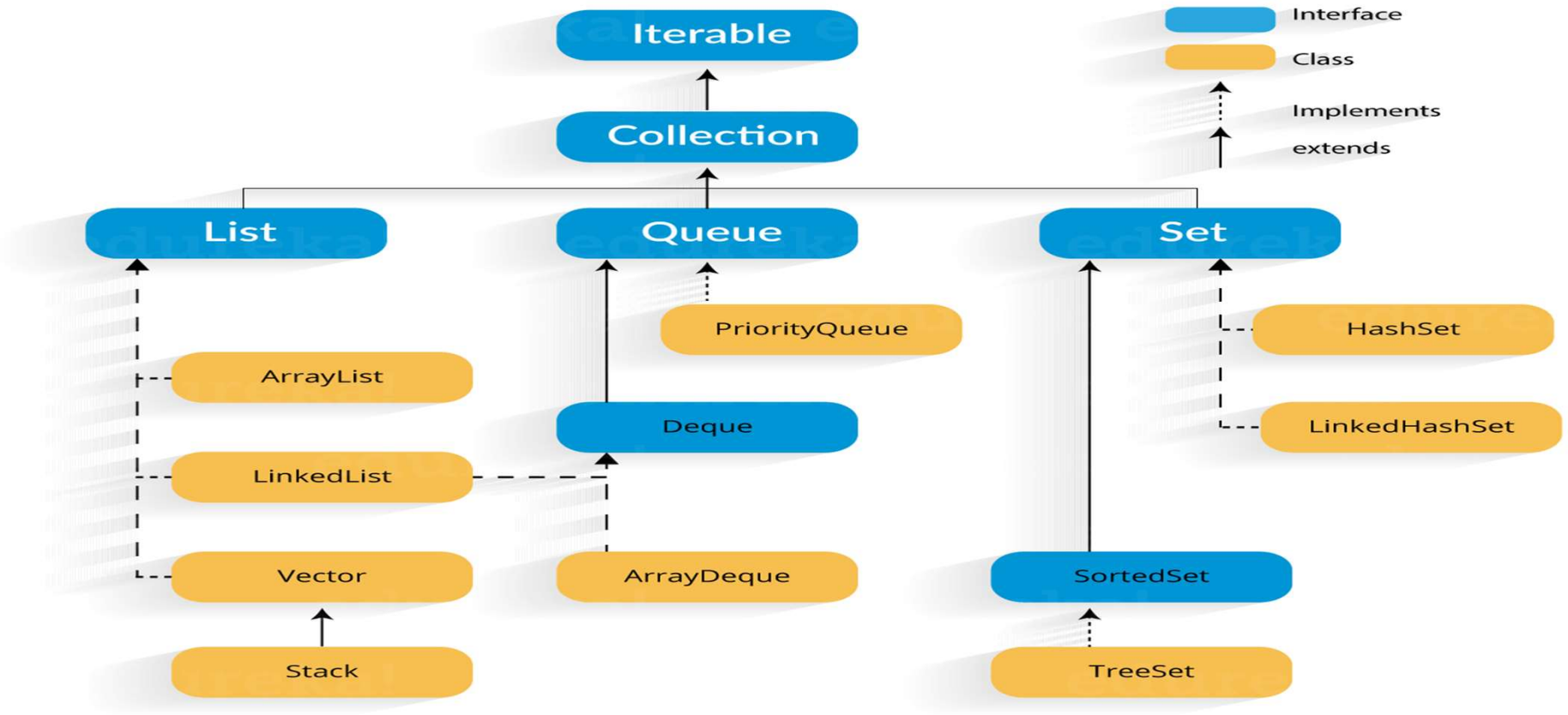




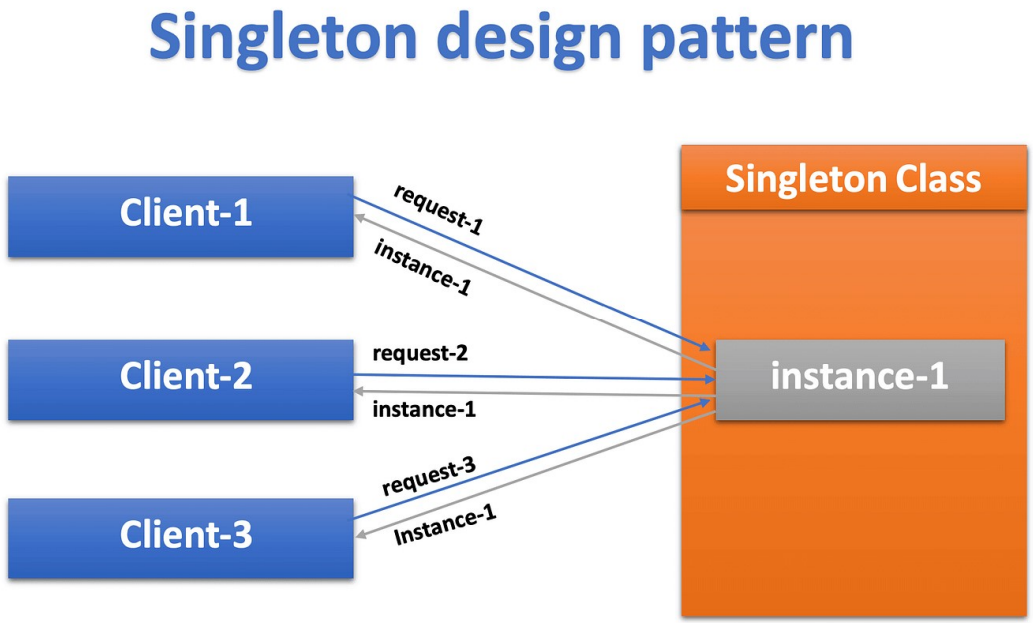
Education that enlightens!

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





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.

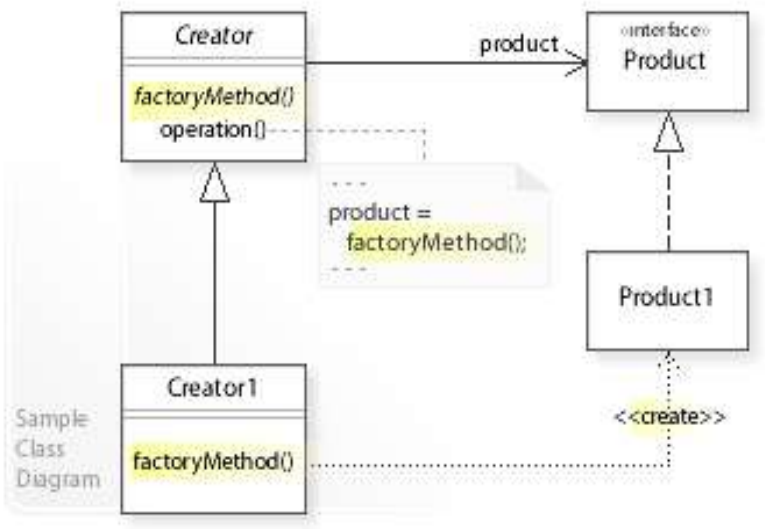


Education that enlightens!

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.



Education that enlightens!

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.).