```
Java Library
============
Java is all about Classes and Objects.

The Java Library contains a lot of predefined
      - classes
      - interfaces
      - enums
      - methods
      - variables

There are 2 types of constructs that we will use:

1. Pre-defined ones
      - There are the many pre-defined classes in Java Library
      - We have to learn to use them
      - String, System, Math are predefined classes

      - print(), println(), pow() are predefined methods

      - out, in, PI are predefined variables

2. User-defined ones
      - These are the classes that we as developers create
      - We have to learn to create and use them
      - Ex01Addition, Q01CircleMeasurement are userdefined classes
      - main() is the user-defined method
      - number1, number2, area, circumference etc are user defined
variables.

Important predefined classes
============================
These are the frequently used classes in Java Library:
1. Object class
2. String Handling classes
3. Wrapper classes
4. Date/Time classes
5. JDBC classes
6. Exception handling classes
7. File Handling classes
8. Collection classes
9. Network classes
10. etc

Object class
============
- Object class is the root of the Java class hierarchy.

- Every class has Object as a superclass automatically.
      1. Whenever we create any class, it implicitly extends the Object
class
      2. All the predefined class automatically extends the Object class
      3. All the classes directly or indirectly extend the Object class
```

```
- It contains following methods
      - Object()
      - clone()
      - equals(Object)
      - finalize()
      - getClass()
      - hashCode()
      - notify()
      - notifyAll()
      - toString()
      - wait()
      - wait(long)
      - wait(long, int)
      - wait0(long)

- All the methods of Object class are available for all its child classes
- As Object class is the parent class of all classes, the methods that
contain Object as a parameter can pass all class objects

Object obj = new Figure();
Object obj = new Student();
Object obj = new Employee();
Object obj = new Rectangle();

If Object is a parameter, we can pass any object

public boolean add(Object obj) {}

Student student = new Student();
add(student);

Employee emp1 = new Employee();
add(emp1);

instanceof
==========
- instanceof operator is used to test whether the object is an instance
of the specified type (class or subclass or interface).
- Whenever we create an instance, it is an instance of its own class and
all its parent classes and interfaces.
- instanceof operator returns true if an object is of its Type

Ex:

class Faculty {}
class FullTimeFaculty extends Faculty {}

FullTimeFaculty fullTimeFaculty = new FullTimeFaculty();

fullTimeFaculty is an instance of FullTimeFaculty
fullTimeFaculty is an instance of Faculty also
fullTimeFaculty is an instance of Object also
```

```
Important: instanceof works only on objects that are in single/multilevel
inheritance, otherwise it will give compile time error

instanceof returns
- true when the object being tested is an instance of the specified
class, its subclass, or an implementation of an interface.
- false when the object is not an instance of the specified class,
subclass, or implementing class, or when the object is null.

instanceof gives
- an error when there is no possible relationship between the object and
the type being checked, resulting in incompatible types. This is a
compile-time error.

Fully Qualified Name
---------------------
name of the package + name of the class
Ex: ex01.object_class + Height => ex01.object_class.Height

- We can have multiple classes in Java by the same name.
Ex: java.sql.Date, java.util.Date
- Class names are not unique
- Fully Qualified names are unique

getClass()
----------
getClass() returns the runtime class of this Object.

Height h1 = new Height();

System.out.println(h1.getClass()); // class ex01.object_class.Height
System.out.println(h1.getClass().getName()); // ex01.object_class.Height

Hashing
-------
Hashing is a mechanism to encrypt data.
There are various hashing alogorithms available.
Ex: MD5, SHA-256, SHA-512...

Ex:
Hi   => 49f68a5c8493ec2c0bf489821c21fc3b
H i  => 4afa9cbce065ea795f42922bf84cd0d6
H    => 2510c39011c5be704182423e3a695e91

hashCode()
----------
- It returns a hash code value for the object.
- This hash code is a unique number used to identify an object in memory
- Hashing is a process by which Java uses to identify an object uniquely
in memory

toString()
----------
```

```
- Whenever we write an object inside print() method, it implicitly calls
the toString() of object class.

System.out.println(product);
means
System.out.println(product.toString());

- toString() returns : fully qualified name of the class + @ +
Hexadecimal String

* Hexadecimal string is obtained by converting hash code of the object to
hex.

Data Representation
===================
class Student {
    int id;
    String name;
    String[] subjects;
}

Object Representation
---------------------
String[] subjects = {"Math", "Science", "History"};
Student student = new Student(1, "Alice", subjects);

String Represenation
--------------------
Student ID: 1, Name: Alice, Subjects: Math, Science, History

JSON Representation
--------------------
{
    "id": 1,
    "name": "Alice",
    "subjects": ["Math", "Science", "History"]
}

XML Represenation
-----------------
<Student>
    <id>1</id>
    <name>Alice</name>
    <subjects>
        <subject>Math</subject>
        <subject>Science</subject>
        <subject>History</subject>
    </subjects>
</Student>

equals
======
Shallow Comparison
------------------
```

```
- Whenever we compare the references, its known as Shallow comparison
- In Shallow Comparison we compare the adresses of objects
Ex: Height h1 = new Height(5, 10), Height h2 = new Height(5, 10)

- The equal to operator (==) does shallow comparison
Ex: h1 == h2 will return false

- The default implementation of equals() also does shallow comparison
Ex: h1.equals(h2) will return false

Deep Comparison
---------------
- When we compare the values in object fields, it is deep comparison.
- We can override equals() in Object class to do deep comparison
-----X-----X-----X-----X-----
Passing and Returning Objects
-----------------------------
Ex: Height totalHeight = h1.add(h2)

String Handling
===============
Inorder to handle String data type, we have 3 main classes:
1. String class
2. StringBuilder class
3. StringBuffer class

Many Helper classes:
1. StringJoiner
2. StringTokenizer
3. etc

String
======
1. Strings are Immutable for memory conservation.
2. + operator is the only operator that is overloaded for String
3. String name = "Sachin";

Note:
- Any sequence of characters enclosed within double quotes is by default
of type String.
For ex: System.out.println("Hello Java");
Here the type of "Hello Java" is string.

- We can call any String class methods on a String literal
Ex 1: "Hello Java".substring(6) will return "Java"
Ex 2: "Hello Java".length() will return 10

String v/s StringBuilder v/s StringBuffer
=========================================
If we modify a String in any way, it will create a new String object.
It is a bad practice to use String class for strings that require many
manipulations.

For multiple manipulation on strings, we use StringBuilder
```

StringBuffer is thread safe, it should be used to create synchronized application

String is used for simple data storage and data transfer

How to use a method?

Whenever we use any method, take into consideration following 4 things:
1. What type and number of parameters we have to pass
2. We should know the significance of parameters
3. What the method returns
4. The exception(s) that the method throws in case of wrong input

Frequently used mathods for class String:

1. length() - Returns the string's length.
2. charAt(int index) - Retrieves character at specified index.
3. substring(int beginIndex, int endIndex) - Extracts part of a string.
4. indexOf(String str) - Finds index of substring occurrence.
5. toLowerCase() - Converts string to lowercase characters.

StringJoiner class
==
- The StringJoiner class in Java is used to construct strings with a delimiter, a prefix, and a suffix.
- It's initialized using the constructor: StringJoiner(CharSequence delimiter, CharSequence prefix, CharSequence suffix).
- Strings are added using the add(CharSequence element) method.
- Multiple StringJoiner instances can be merged using the merge(StringJoiner other) method.

StringTokenizer class
==
- The StringTokenizer class in Java is used to break a string into tokens or parts based on a specified delimiter.
- It's initialized using the constructor StringTokenizer(String str, String delim).
- It contains methods like nextToken(), countTokens(), split() to work with strings

Wrapper Classes
==
- Java, as we know, is primarily an object oriented language.
- What it means is : it contains many classes and methods exclusively designed to support objects.
- The existence of primitive types is not in accordance with the OOP philosophy.
- For example: Collection framework does not work with primitive values
  List<int> numbers will give CTE
- In order to represent primitive types as objects, we use Wrapper classes.
- We have 8 Wrapper classes, 1 each for each primitive type.

```
Primitive type - Wrapper Type
byte   - Byte
short  - Short
int    - Integer
long   - Long
float  - Float
double - Double
bool   - Boolean
char   - Character

- We can type cast double to int in case of primitives, but we can't do
it in case of objects as they are sibling classes.
- The small to big hierarchy does not exist in Wrapper classes
- All numerical classes extend the Number class and are siblings of each
other
- All wrapper classes are independent classes.

Frequently used mathods for class Integer:

1. parseInt(String s) - Converts string to integer value.
2. valueOf(String s) - Returns wrapped object from string.
3. intValue() - Converts object to primitive int.
4. compareTo(Object o) - Compares two objects for order.

Similar methods are there for other Wrapper classes.
-----X-----X-----X-----X-----
Date Time Classes
=================
- Before Java 8, we had java.util.Date and java.util.Calendar classes
- In Java 8, new date/time classes were added in java.time package
- Some of the most used classes are :
      - LocalDate
      - LocalTime
      - LocaldateTime
      - Period
      - Duration

- Some of the common methods of these classes are: now(), parse(), of(),
compare(). equals()

- The standard format used to represent dates is : "YYYY-MM-dd"
- This format is known as ISO format

Some frequently used methods of the LocalDate class:

1. now() - Gets the current date.
2. of(int year, int month, int day) - Creates a date with specified
values.
3. plusDays(long days) - Adds days to current date.
4. minusMonths(long months) - Subtracts months from current date.
5. isBefore(LocalDate otherDate) - Checks if date is before another.

-----X-----X-----X-----X-----
```

```
JDBC API
========
- The JDBC API consists of classes and interfaces used to connect and
transact with the underlying database.
- Ex: Connection, DriverManager, Statement, PreparedStatement, ResultSet
- We will dig them up in JDBC.
-----X-----X-----X-----X-----X-----
```