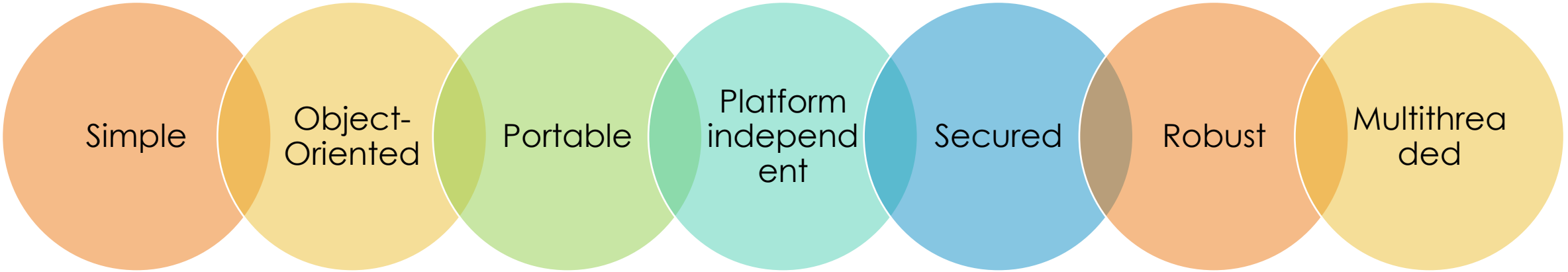
INTRODUCTION TO

A Powerful Programming Language

JAVA BASICS

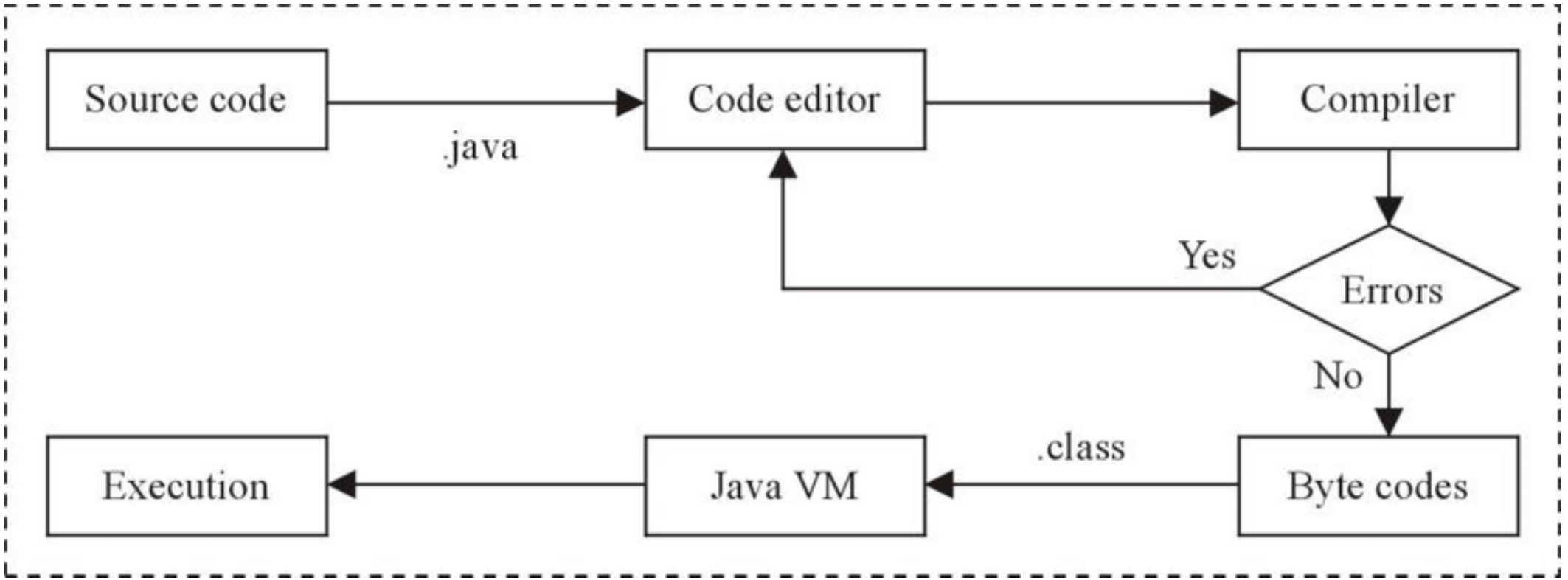
- Java is one of the most popular and widely used programming language and platform.
- A platform is an environment that helps to develop and run programs written in any programming language.
- **Java** is object-oriented programming language.
- It is intended to let application developers ***write once, run anywhere* (WORA)**,
- meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.

JAVA FEATURES

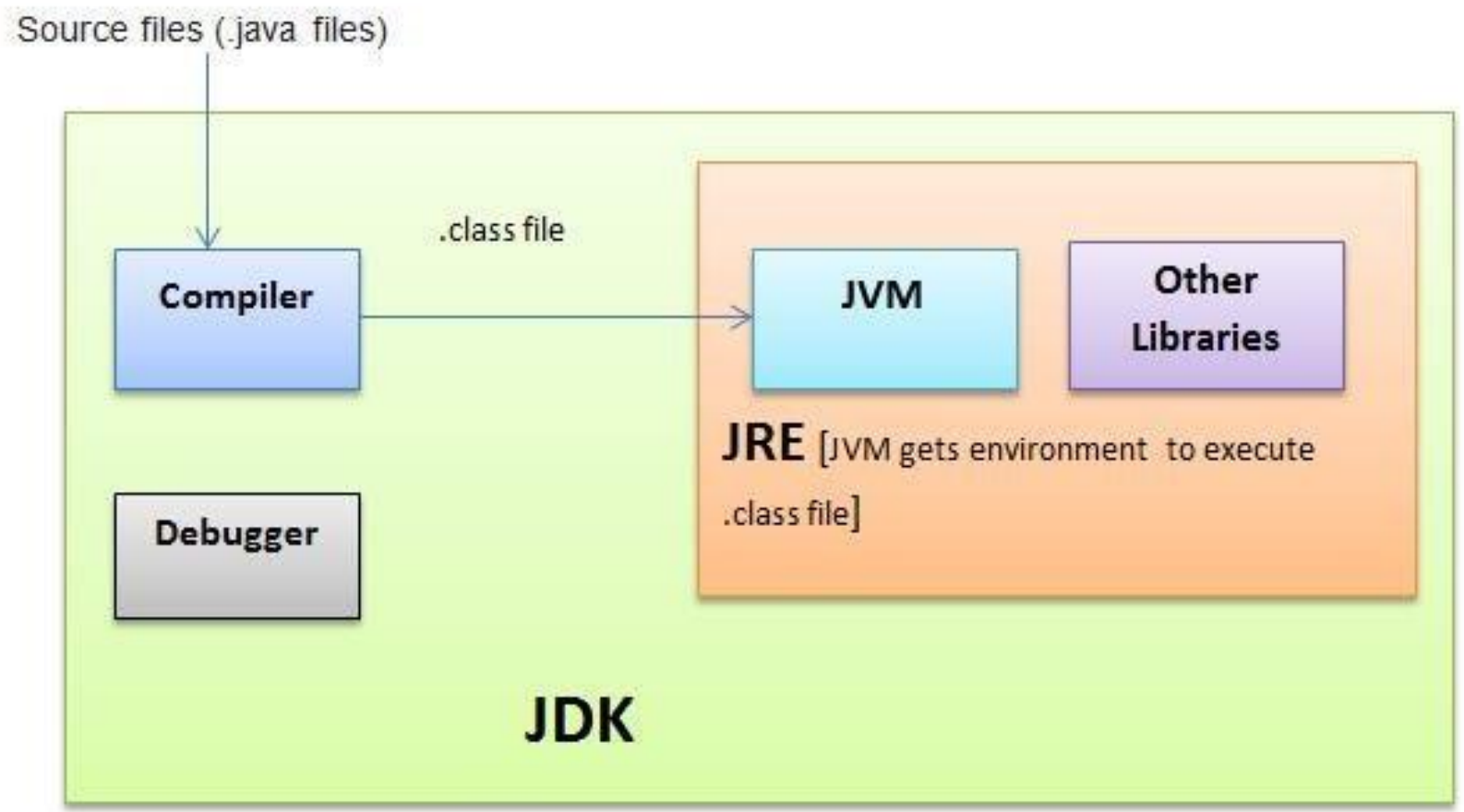


JAVA PLATFORM INDEPENDENT

- **Platform independent** language means once compiled you can execute the program on any **platform** (**OS**).
- The Java compiler converts the source code to bytecode, which is an intermediate Language.
- The bytecode generated is a non-executable code and needs an interpreter to execute on a machine. This interpreter is the JVM and thus the Bytecode is executed by the JVM.
- **Java is platform independent but JVM is platform dependent.**


Source code
.java
Code editor
Compiler
Yes
Errors
No
Execution
Java VM
.class
Byte codes

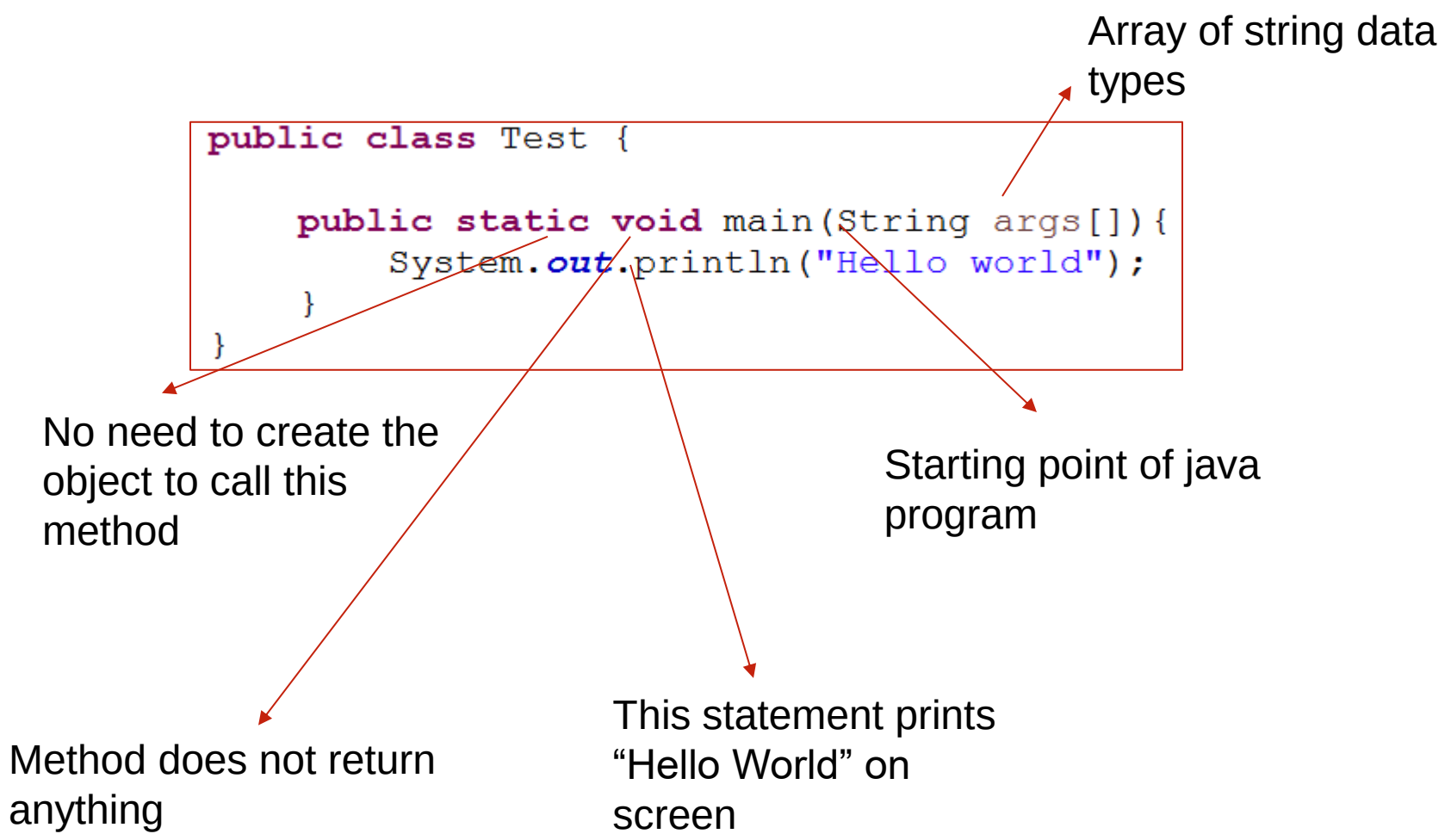
JDK, JRE AND JVM



JDK, JRE, JVM

- JDK (Java Development Kit) is a software development kit required to develop applications in Java. When you download JDK, JRE is also downloaded with it.
- In addition to JRE, JDK also contains a number of development tools (compilers, JavaDoc, Java Debugger, etc).
- JRE (Java Runtime Environment) is a software package that provides Java class libraries, Java Virtual Machine (JVM), and other components that are required to run Java applications.
- JRE is the superset of JVM.
- JVM (Java Virtual Machine) is an abstract machine that enables your computer to run a Java program.
- When you run the Java program, Java compiler first compiles your Java code to bytecode. Then, the JVM translates bytecode into native machine code (set of instructions that a computer's CPU executes directly).

HELLO WORLD PROGRAM



VARIABLES

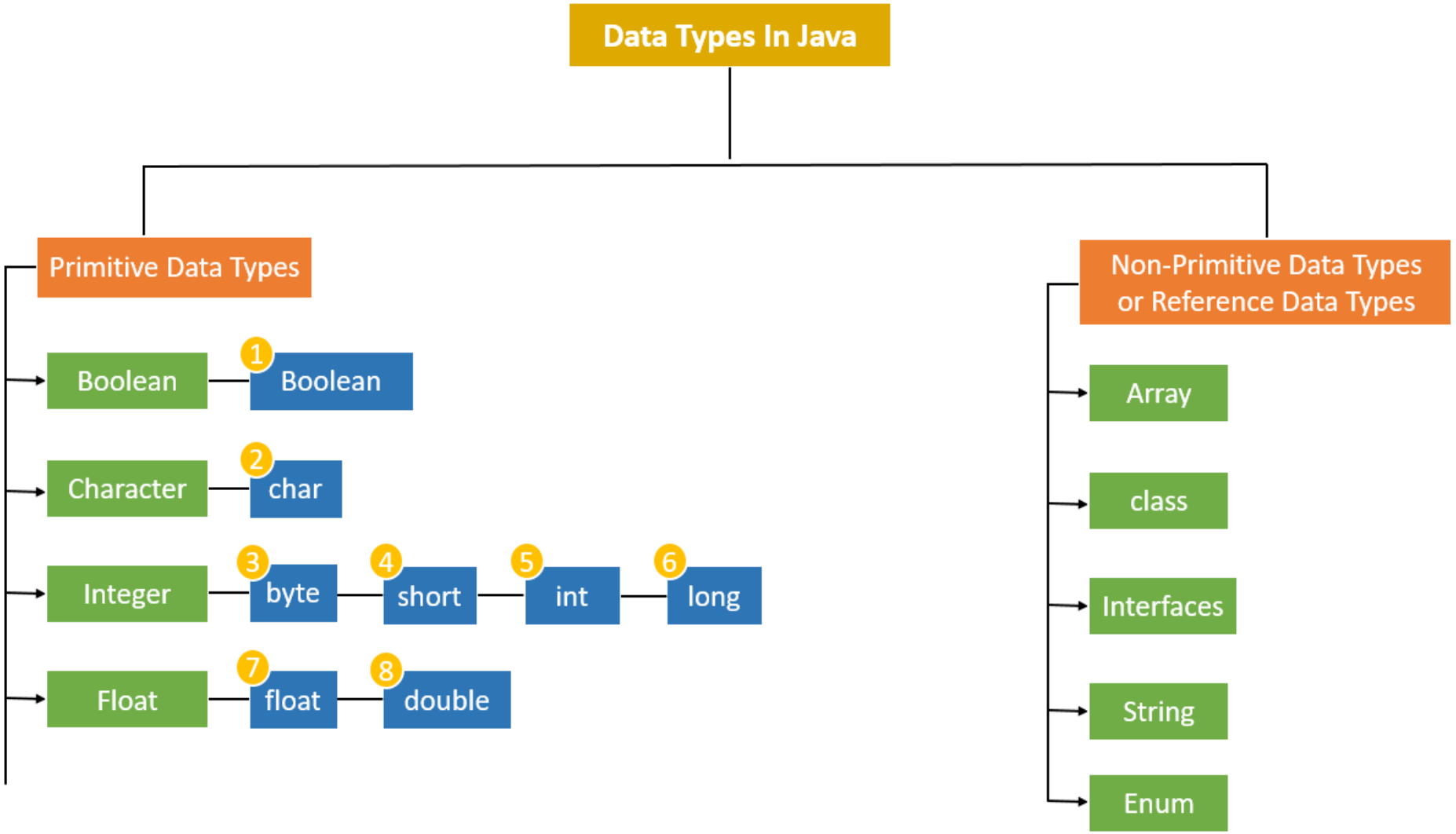
- A variable in Java is a container that holds data that can be changed during the execution of a program. Variables have three main components:
- **Type:** Specifies the type of data the variable can hold.
- **Name:** The identifier used to refer to the variable.
- **Value:** The data held by the variable.

Variable declaration:

```
int age;
String name;
double salary;
```

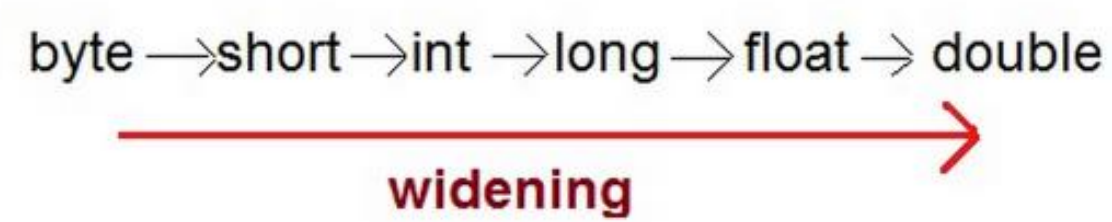
Variable initialization:

```
int age = 25;
String name = "John";
double salary = 50000.0;
```


Data Types In Java
Primitive Data Types
Boolean
1
Boolean
Character
2
char
Integer
3
byte
4
short
5
int
6
long
Float
7
float
8
double
Non-Primitive Data Types or Reference Data Types
Array
class
Interfaces
String
Enum

TYPECASTING

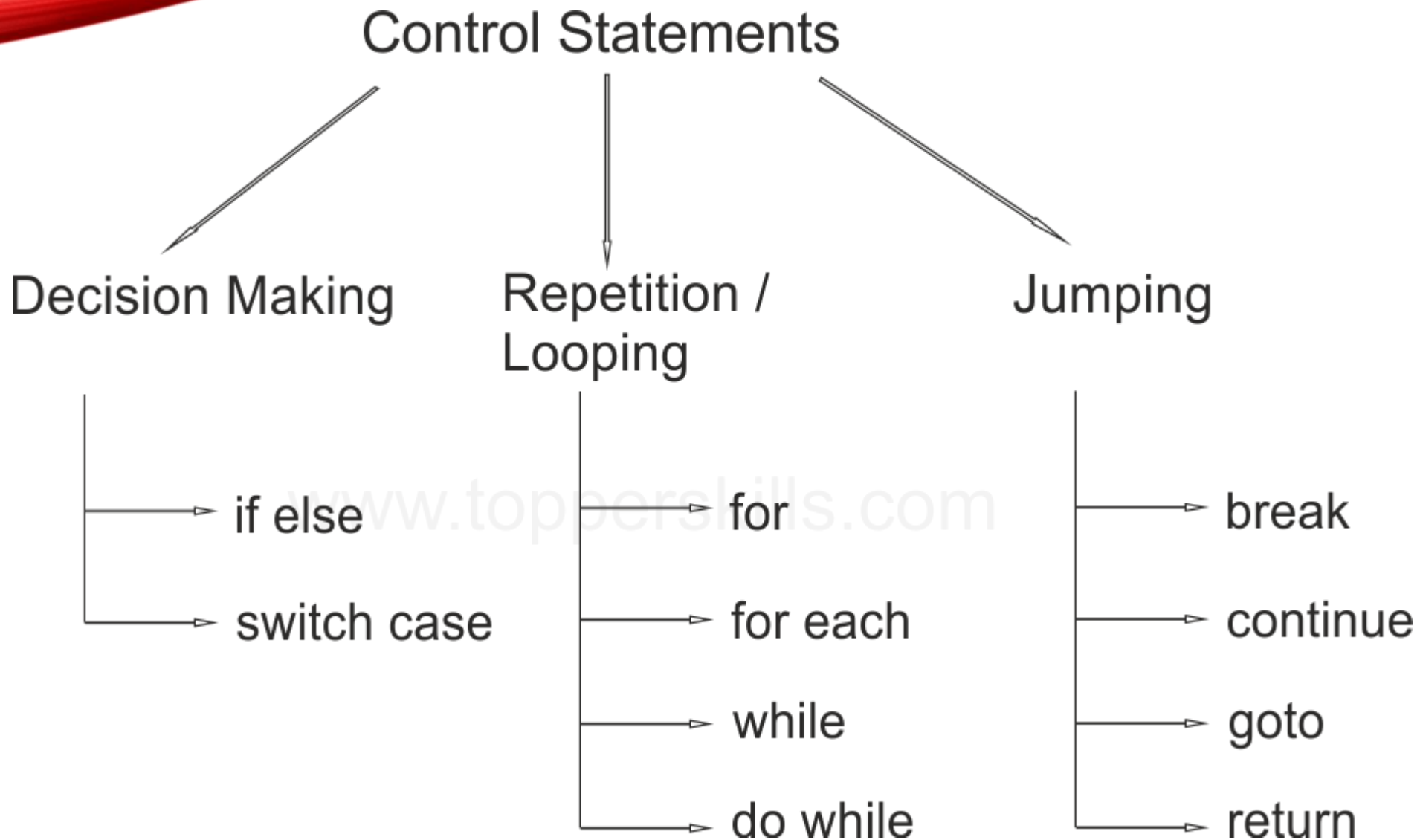
- Widening Casting(Implicit)



- Narrowing Casting(Explicitly done)

OPERATORS

1. Arithmetic Operators : +, -, *, /, %
2. Assignment Operators : =, +=, -=, *=, /=, %=, &=, |=, ^=
3. Relational Operators : >, <, ==, >= , <= , !=
4. Unary Operator : ++, --, !
5. Logical Operators : &&, | |, !
6. Bitwise Operator : &, |, =, ^, >>, <<
7. Ternary Operator: ?:



ARRAY

- Array in Java are a group of like-typed variables that are referred to by a common name. Array is a collection of homogenous elements that have **contiguous** memory location.
- The **size** of an array must be specified by an **int** value and not long or short.
- Array can contain primitive data types as well as objects of a class depending on the definition of array.
- In case of primitive data types, the actual values are stored in **contiguous** memory locations.
- In case of objects of a class, the actual objects are stored in heap area.

ARRAY

- Creating and using arrays in Java involves several steps: declaration, instantiation, initialization, and usage.
- Array Declaration:

```
int[] numbers;
String[] names;
double[] values;
```

- Instantiate an Array:

```
numbers = new int[5];
names = new String[3];
values = new double[4];
```

WORKING WITH ARRAY

```
numbers[0] = 1;
numbers[1] = 2;
numbers[2] = 3;

names[0] = "John";
names[1] = "Jane";
names[2] = "Doe";

values[0] = 1.1;
values[1] = 2.2;
values[2] = 3.3;
```

Access Data in variables:

```
int firstNumber = numbers[0];
String firstName = names[0];
double firstValue = values[0];
```

Access Data in variables:

```
int firstNumber = numbers[0];
String firstName = names[0];
double firstValue = values[0];
```

LOOP THROUGH AN ARRAY

```
for (int i = 0; i < numbers.length; i++) {
    System.out.println("Element at index " + i + ": " + numbers[i]);
}
```

```
for (String name : names) {
    System.out.println("Name: " + name);
}
```

Enhanced For Loop

```
int length = numbers.length;  // Get the length of the array
System.out.println("Length of numbers array: " + length);
```

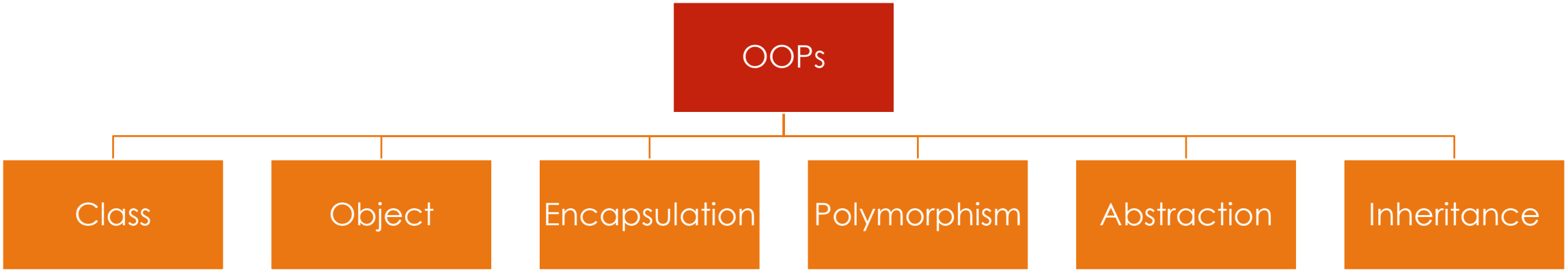
2D ARRAY

```
public class TwoDArray {

    public static void main(String[] args) {

        //int arr[][]=new int[3][3];
        int arr[][]= {{1,2,3},{2,3},{6,7,9,10}};

        System.out.println("Print Array");
        for(int row=0;row<arr.length;row++) {
            for(int col=0;col<arr[row].length;col++) {
                System.out.print(arr[row][col]+"\t");
            }
            System.out.println();
        }
        System.out.println("Print using for each loop");
        for(int row[]:arr) {
            for(int col:row) {
                System.out.print(col+"\t");
            }
            System.out.println();
        }
    }
}
```

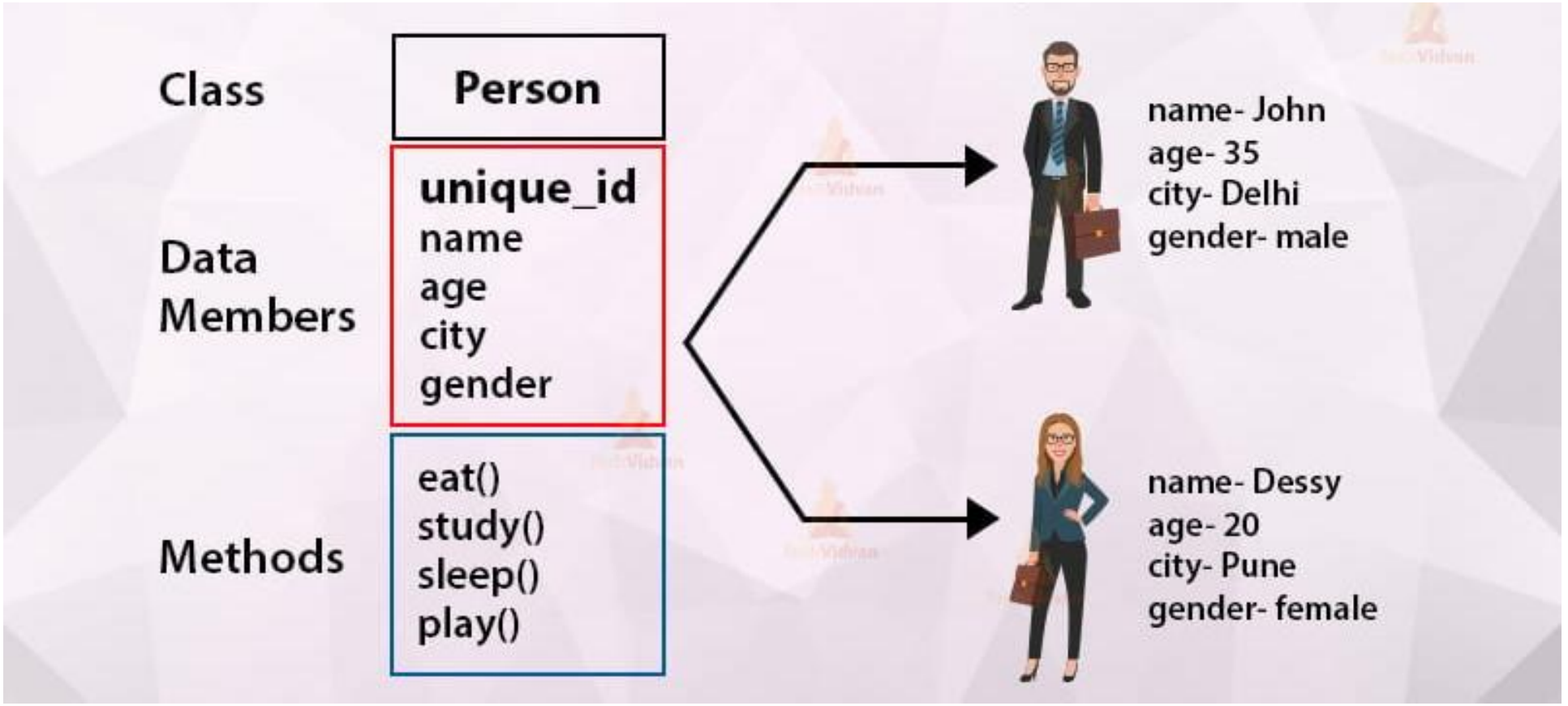
ACTIVITY: USE SCANNER FOR ALL

- Write a program to display table of 10.
- Write a program to display Fibonacci series of 10 numbers (0,0,1,1,2,3,5,8,13,21).
- Write a program to check whether a given number is odd or even.
- Write a program to find largest of two numbers.
- Write a java program to create an array of numbers and display the original array in sorted order.
- Write a java program to create an array of numbers and display the original array in reverse order.

OOPS


OOPs
Class
Object
Encapsulation
Polymorphism
Abstraction
Inheritance

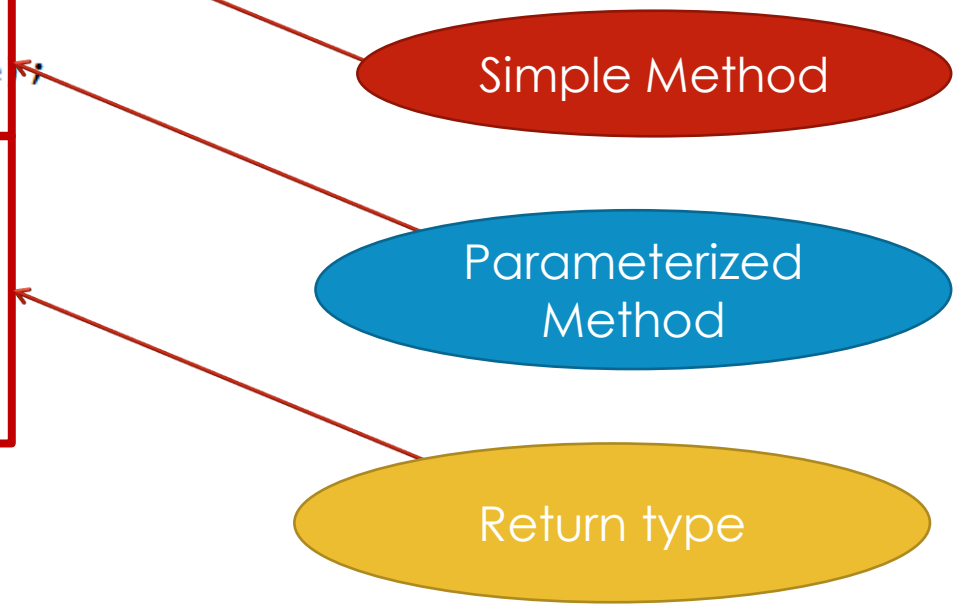
CLASS & OBJECT



```java
public class Student {

    // State
    int id;
    String name;
    String email;
    String address;
    // Behavior
    public void setData(int i, String n, String e, String a) {
        id = i;
        name = n;
        email = e;
        address = a;
    }
    public void speak() {
        System.out.println(name + " is speaking");
    }
    public void eat() {
        System.out.println(name + " is eating");
    }
}
```

METHODS

```
public class Methods1 {

    public void print() {
        System.out.println("Print Method Called");
    }
    public void fullName(String fname,String lname) {
        System.out.println("Welcome "+fname+" "+lname);
    }
    public int cube(int n) {
        return n*n*n;
    }
    public boolean checkValidity(int age) {
        return age>=18;
    }

    public static void main(String[] args) {
        Methods1 obj= new Methods1();
        obj.print();
        obj.fullName("Sonam", "soni");
        System.out.println("Cube of: "+obj.cube(4));
        System.out.println("User Validity "+obj.checkValidity(34));
    }
}
```



CONSTRUCTOR

- Constructors are used to initialize the object's state. Like methods, a constructor also contains **collection of statements** that are executed at time of Object creation.
- Constructors are used to assign values to the instance variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).
- Each time an object is created using **new()** keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the **data members** of the same class.
- There are two type of constructor in Java:
  - No-argument constructor : A constructor is called "Default Constructor" when it doesn't have any parameter.
  - Parametrized constructor : A constructor which has a specific number of parameters is called a parameterized constructor.

DEFAULT & PARAMETRIZED CONSTRUCTOR

```java
class Employee {
    /* instance varibale */
    int id;
    String name;
    float salary;
    void display() {
        System.out.println(this.id + " " + this.name + " " + this.salary);
    }
}
public class Test {
    public static void main(String args[]) {
        //Object creation using default constructor
        Employee emp1= new Employee();
        emp1.display();

    }
}
```

```java
class Employee {
    /* instance varibale */
    int id;
    String name;
    float salary;
    // Parametrized constructor
    Employee(int id, String name, float salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    void display() {
        System.out.println(this.id + " " + this.name + " " + this.salary);
    }
}
public class Test {
    public static void main(String args[]) {
        //Object creation
        Employee emp1 = new Employee(101,"John",20000);
        emp1.display();
        Employee emp2 = new Employee(102,"Jack",10000);
        emp2.display();
    }
```

CONSTRUCTOR VS METHOD

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

ACTIVITY: STUDENT GRADES MANAGEMENT SYSTEM

- create a Student class with the following attributes:
  - name (String)
  - grades (an array of integers representing the grades of the student)
- Implement the following methods:
  - A constructor that initializes the student's name and grades.
  - calculateAverage() method that returns the average grade of the student.
  - displayInfo() method that prints the student's name and their average grade.

THIS KEYWORD

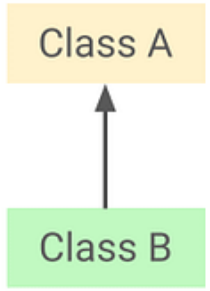
1. this can be used to refer current class instance variable.
2. this() can be used to invoke current class constructor.
3. this can be used to invoke current class method (implicitly)

```
// Parametrized constructor
Employee(int id, String name) {
    this.id = id;
    this.name = name;
}

// Parametrized constructor
Employee(int id, String name, float salary) {
    this(id,name);
    this.salary = salary;
}

void first() {
    System.out.println(this.id + " " + this.name);
    this.second();
}

void second() {
    System.out.println(this.salary);
}
```

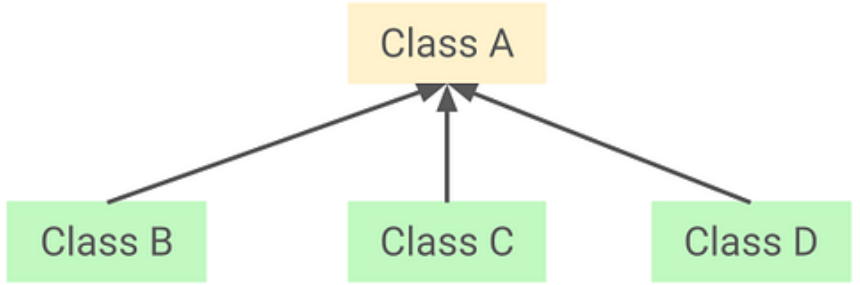
STATIC KEYWORD

- It is used to initialize variable once.
- Also used to call method directly without creating an object of class.
- Actually static variable is creating class level variables and methods.

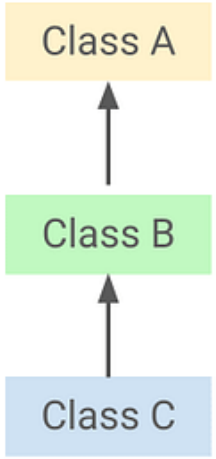
INHERITANCE



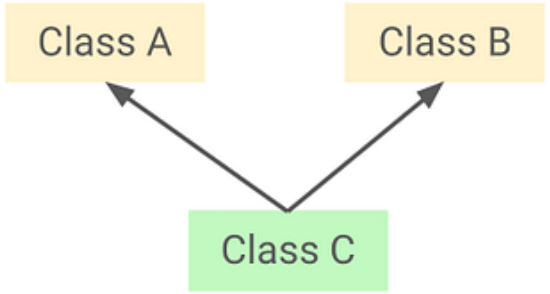
Single Inheritance



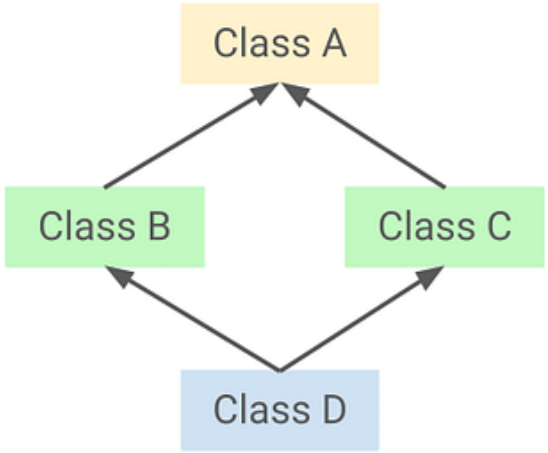
Hierarchical inheritance



Multilevel Inheritance



Multiple Inheritance

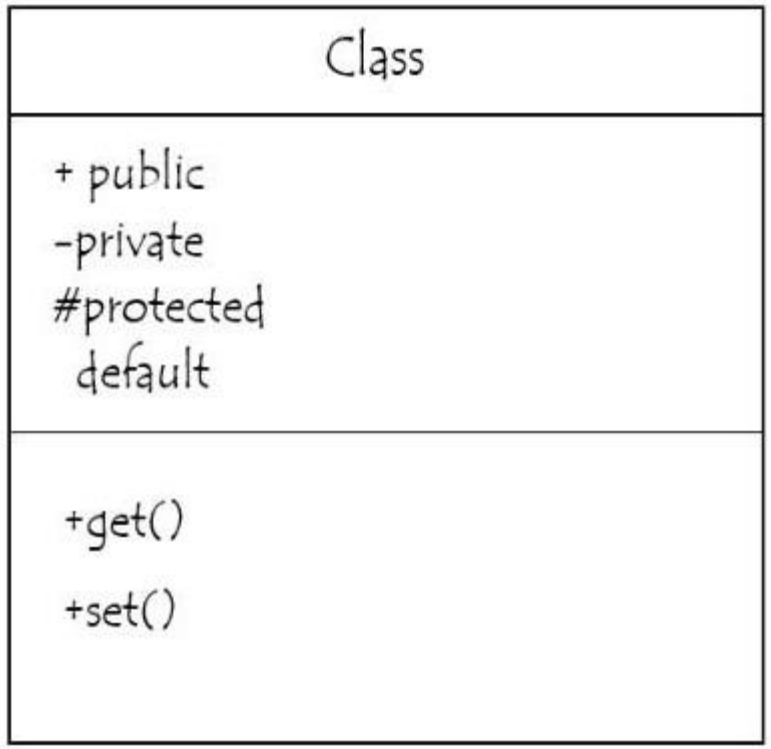


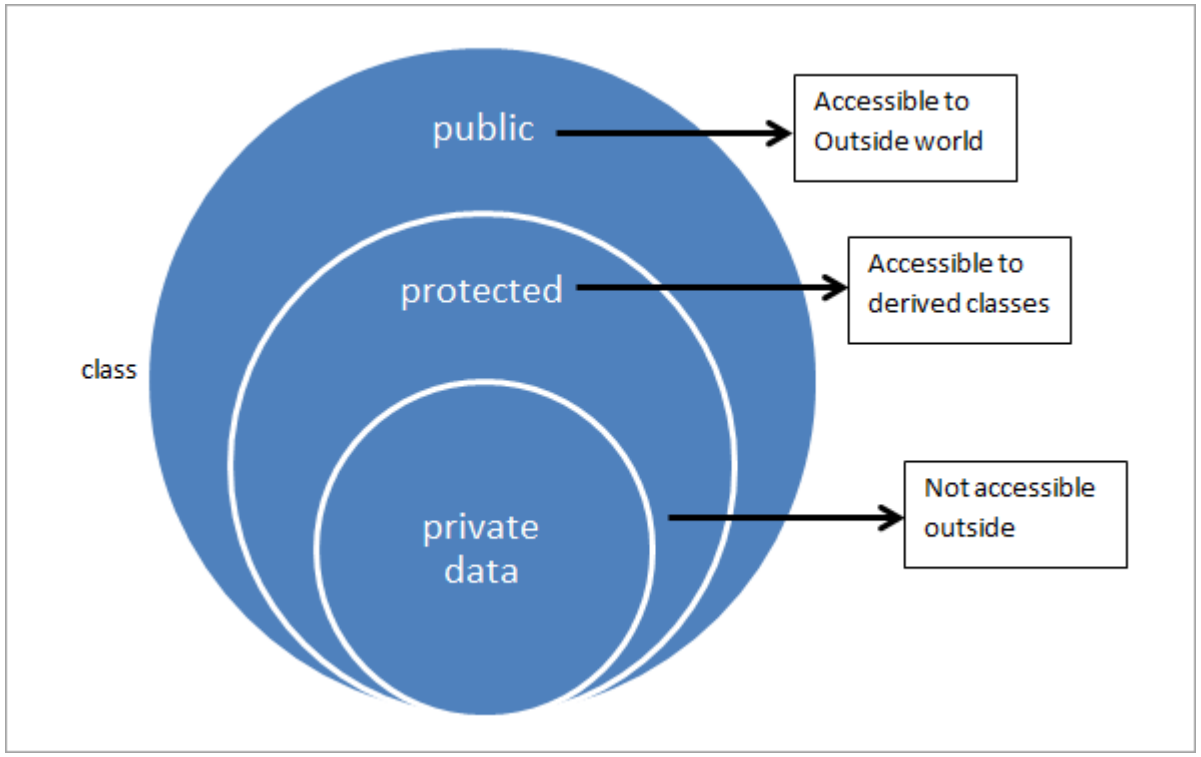
Hybrid Inheritance

SUPER KEYWORD

- It is used to call parent class constructor
- Used to pass data from parent to child.
- Used to call methods from child to parent.
  - Super.parentclassmethod()

ENCAPSULATION





LETS IMPLEMENT IT

- Make all the customer fields as private. Write getter and setters for customer class for all fields.
- Create customer objects using setter methods.
- Display the array of customer objects.

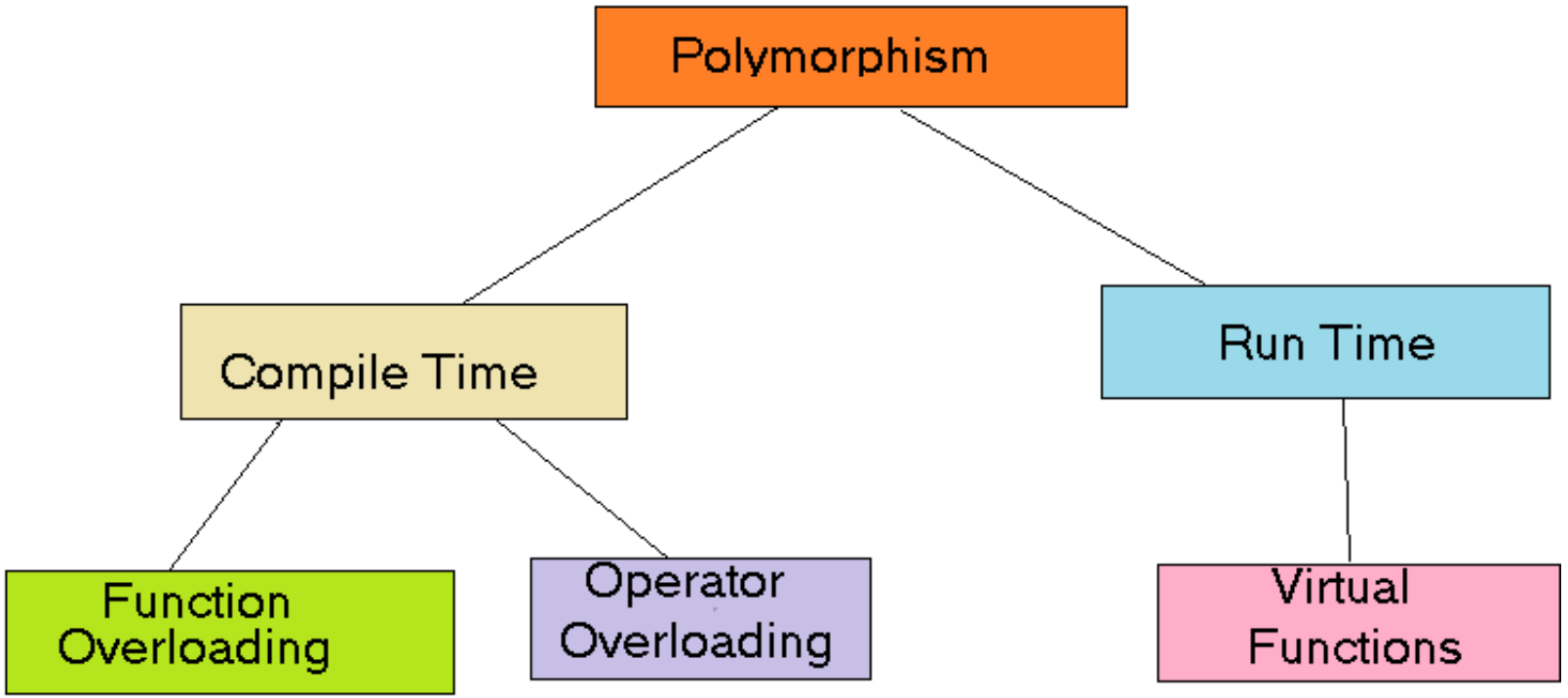
ABSTRACTION

- An abstract class is a class that is declared with abstract keyword.
- An abstract class can never be instantiated. If a class is declared as abstract then the sole purpose is for the class to be extended.
- A class cannot be both abstract and final. (since a final class cannot be extended).
- An abstract method is a method that is declared without an implementation.

DATA ABSTRACTION

- Implemented in 2 ways
  - Abstract class & Abstract methods
  - Interface

POLYMORPHISM



FINAL KEYWORDS

- We can create final variables which is constant.
- We can create final method which can not be overridden.
- We can create final class which can not be inherited.

STRING

- String is an object that represents a sequence of characters.
- The java.lang.String class is used to create a string object. String objects are immutable (Once you create you can not change it.)
- There are two ways to create String object:
  - By string literal:

    String s="Welcome"; (creates one string in constant pool area if not found else it will point to same string)
  - By new keyword:

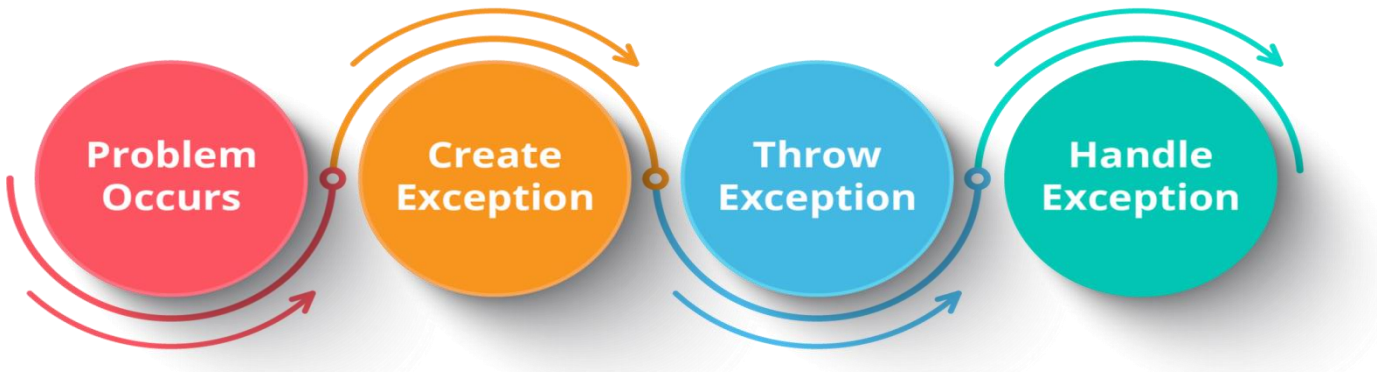
    String s= new String("Welcome"); (creates one string in constant pool area and one object in heap area)

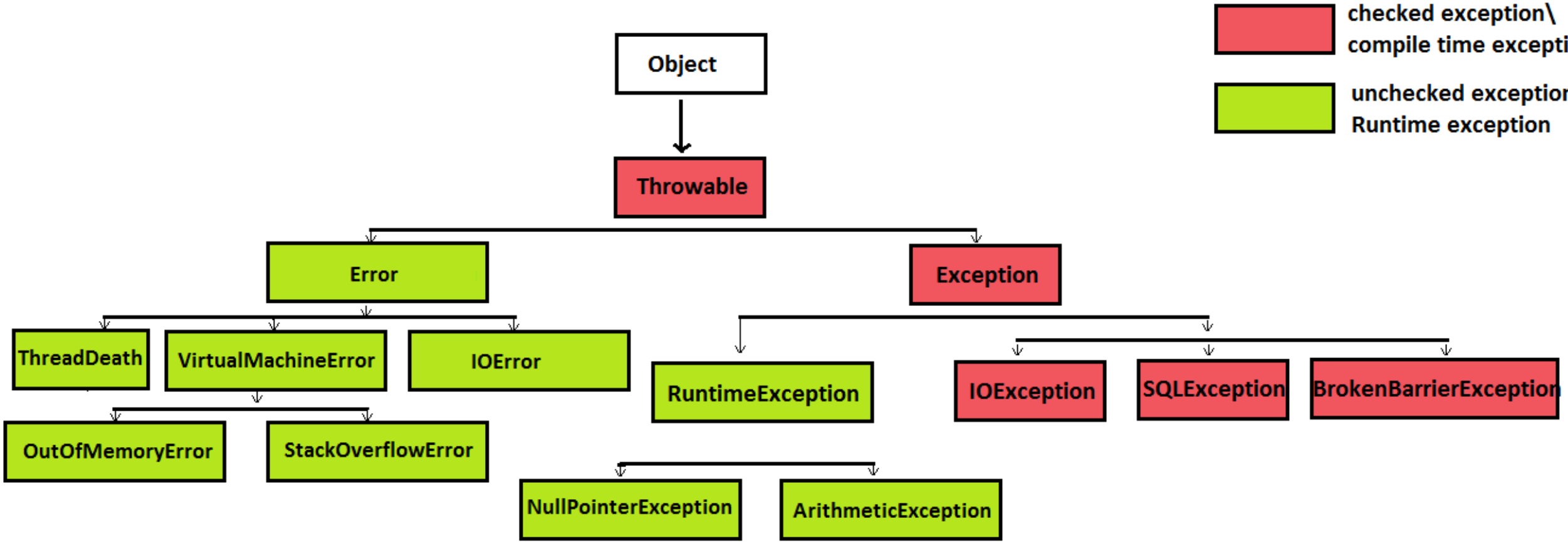
STRING IMMUTABILITY

- To create mutable string we can use:
- String Buffer
- String Builder

EXCEPTION HANDLING

- **What is an Exception?**
  - An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time, that disrupts the normal flow of the program's instructions. It is an **object** which is thrown at **runtime**.
- **What is exception handling?**
  - Exception Handling is a mechanism to handle runtime errors such as Arithmetic, ClassNotFound, IO, SQL etc. So it is a way to provide a proper structure when an exception occurs such that the program execution is not affected.
  - Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmer's fault that he is not performing check up before the code being used.

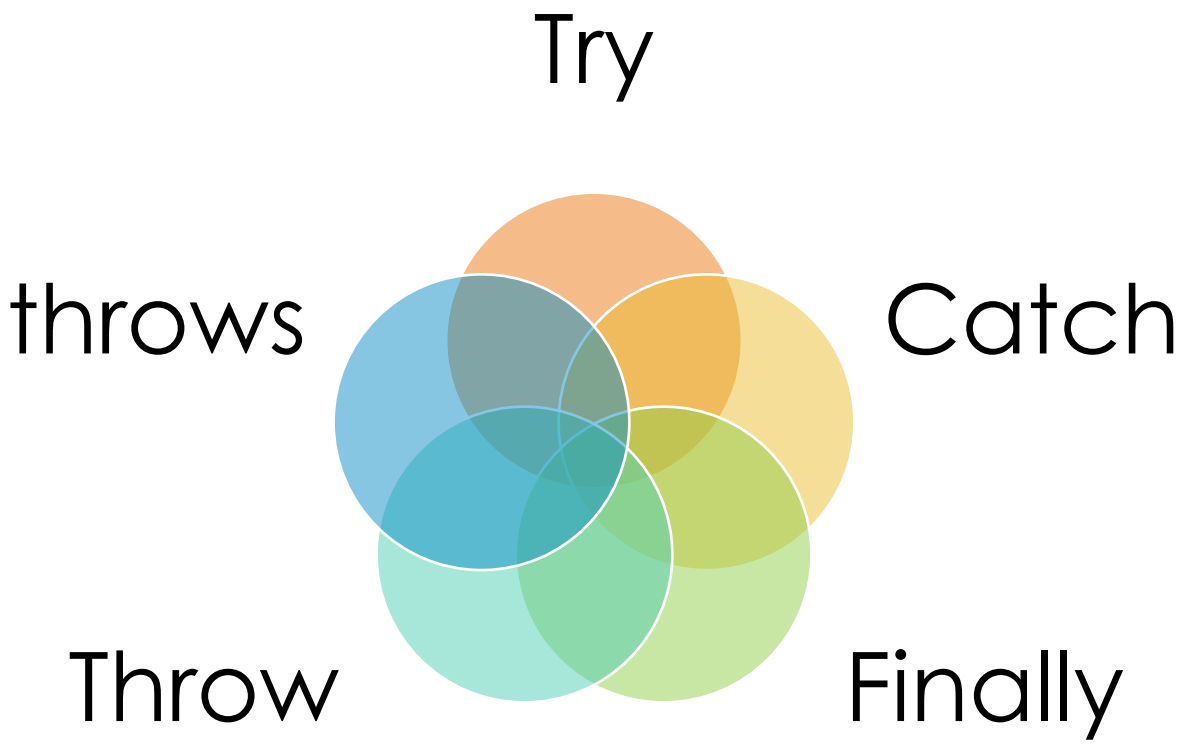



checked exception\
compile time exception
unchecked exception\
Runtime exception
Object
Throwable
Error
Exception
ThreadDeath
VirtualMachineError
IOError
RuntimeException
IOException
SQLException
BrokenBarrierException
OutOfMemoryError
StackOverflowError
NullPointerException
ArithmeticException

EXCEPTION

- Checked Exception:
  - Occurs at compile time
  - Checked by the compile
  - We have to handle it
- Unchecked Exception:
  - Occurs at runtime
  - Checked at the dynamic (runtime) execution
  - If we will not handle program executed but if it throws error then program terminates.

HANDLE THE EXCEPTION



ADVANCE TOPICS ON EXCEPTION

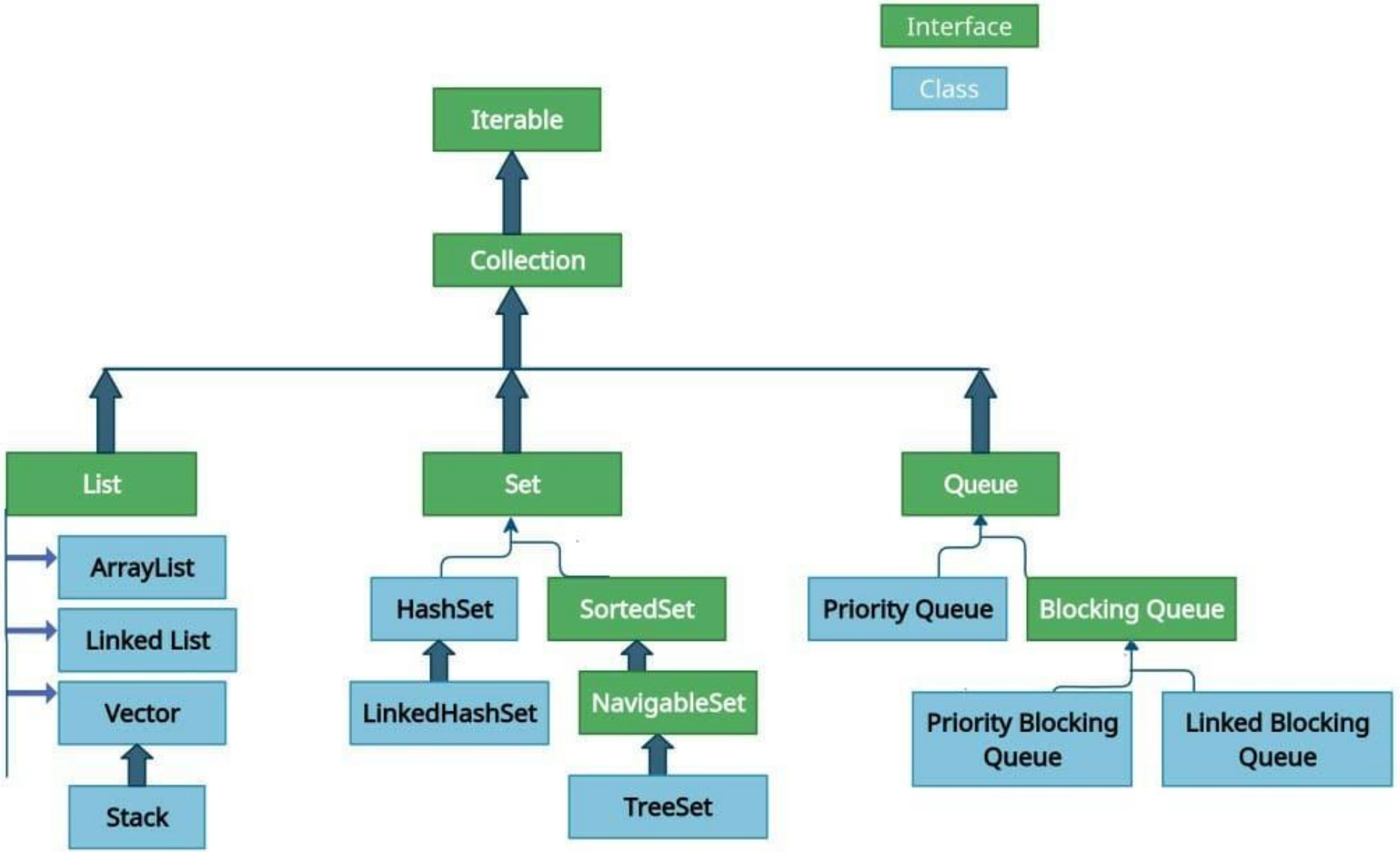
- Multi catch blocks
- Exception propagation (chaining)
- Custom Exception

WRAPPER CLASS

Primitive Data Type	Wrapper Class
char	Character
byte	Byte
short	Short
long	Integer
float	Float
double	Double
boolean	Boolean

COLLECTIONS

- The **Java collections framework** is a set of classes and interfaces that implement commonly reusable collection data structures.
- Although referred to as a framework, it works in a manner of a library. The collections framework provides both interfaces that define various collections and classes that implement them.
- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).


Interface
Class
Iterable
Collection
List
Set
Queue
ArrayList
Linked List
Vector
Stack
HashSet
SortedSet
LinkedHashSet
NavigableSet
TreeSet
Priority Queue
Blocking Queue
Priority Blocking Queue
Linked Blocking Queue

LIST INTERFACE

- The List interface in Java is part of the Java Collections Framework and represents an ordered collection of elements that can contain duplicates.
- It extends the Collection interface and provides methods to manipulate elements based on their position (index).
- List Implementation
  - ArrayList
  - LinkedList
  - Vector
  - Stack

ARRAY LIST & LINKED LIST

- **ArrayList**:
  - Resizable array implementation.
  - Allows random access to elements.
  - Fast iteration and random access.
  - Slower when adding/removing elements in the middle.
  - **When you use:** When frequent access to elements by index is required and insertion/deletion operations are rare.
- **LinkedList:**
  - Doubly-linked list implementation.
  - Allows sequential access.
  - Faster insertion and removal operations in the middle.
  - Slower when accessing elements by index.
  - **When to use:** When frequent insertion/deletion operations are required and access by index is rare.

VECTOR & STACK

- **Vector**:
  - Synchronized version of ArrayList.
  - Thread-safe but with additional overhead.
  - Generally slower than ArrayList due to synchronization.
  - **When to use**: When a thread-safe implementation of a list is required
- **Stack:**
  - Extends Vector to implement a last-in, first-out (LIFO) stack.
  - Includes standard stack operations such as push, pop, and peek.
  - **When to use**: When a stack structure is required.

SET INTERFACE

- The Set interface in Java is a part of the Java Collections Framework and represents a collection that does not allow duplicate elements.
- It models the mathematical set abstraction and is used to store unique elements.
- Its implementation:
  - HashSet
  - LinkedHashSet
  - TreeSet

Feature	ArrayList	LinkedList	Vector	Stack
Underlying Data Structure	Resizable array	Doubly-linked list	Resizable array	Resizable array (inherits from Vector)
Random Access	Fast	Slow	Fast	Fast
Insertion/Deletion (Middle)	Slow	Fast	Slow	Slow
Thread-Safety	No	No	Yes	Yes
Iteration Performance	Fast	Fast (if sequential)	Fast	Fast
Memory Overhead	Less (Array size may need resizing)	More (Node objects)	Less (Array size may need resizing)	Less (Array size may need resizing)
Additional Methods	None	None	Synchronized methods	Stack operations (push, pop, peek)

HASHSET

- **HashSet**
  - Implements a hash table.
  - No guarantees on the iteration order.
  - Allows null elements.
  - Best performance for most operations (add, remove, contains) due to constant time complexity (O(1)).
- **When to Use**: When you need a collection with unique elements and don't care about the order of iteration.

LINKEDHASHSET

- **LinkedHashSet**
  - Extends HashSet and maintains a doubly-linked list across all elements.
  - Guarantees iteration order (insertion order).
  - Allows null elements.
  - Slightly slower than HashSet due to the added cost of maintaining the linked list.
- **When to use**: When you need a collection with unique elements and want to maintain the order of insertion.

TREESET

- **TreeSet**:
  - Implements a balanced tree (Red-Black tree).
  - Guarantees that the elements will be in ascending order (natural order) or according to a provided comparator.
  - Does not allow null elements.
  - Performance is O(log n) for most operations (add, remove, contains).
- **When to use**: When you need a sorted set of unique elements.

Feature	HashSet	LinkedHashSet	TreeSet
Underlying Data Structure	Hash table	Hash table + Linked list	Red-Black tree
Iteration Order	Unordered	Insertion order	Sorted order
Allow null Elements	Yes	Yes	No
Performance (Add, Remove, Contains)	O(1)	O(1) (slightly slower than HashSet)	O(log n)
Memory Overhead	Less	More (due to linked list)	More (due to tree structure)

MAP INTERFACE

- The Map interface in Java represents a collection of key-value pairs where each key maps to exactly one value.
- It does not allow duplicate keys but allows duplicate values. The Map interface is a part of the Java Collections Framework.
- Its Implementation:
  - HashMap
  - LinkedHashMap
  - HashTable
  - TreeMap

HASHMAP

- Implements a hash table.
- No guarantees on the order of the map.
- Allows null values and one null key.
- Offers constant-time performance for the basic operations (add, remove, contains).
- **When to use:** When you need a map with unique keys and do not care about the order.

LINKED HASH MAP

- Extends HashMap and maintains a doubly-linked list across all entries.
- Guarantees iteration order (insertion order).
- Allows null values and one null key.
- Slightly slower than HashMap due to the added cost of maintaining the linked list.
- **When to use**: When you need a map with unique keys and want to maintain the order of insertion.

TREEMAP

- Implements a Red-Black tree.
- Guarantees that the keys will be in ascending order (natural order) or according to a provided comparator.
- Does not allow null keys (but allows null values).
- Performance is O(log n) for most operations (add, remove, contains).
- **When to use**: When you need a sorted map of unique keys.

HASH TABLE

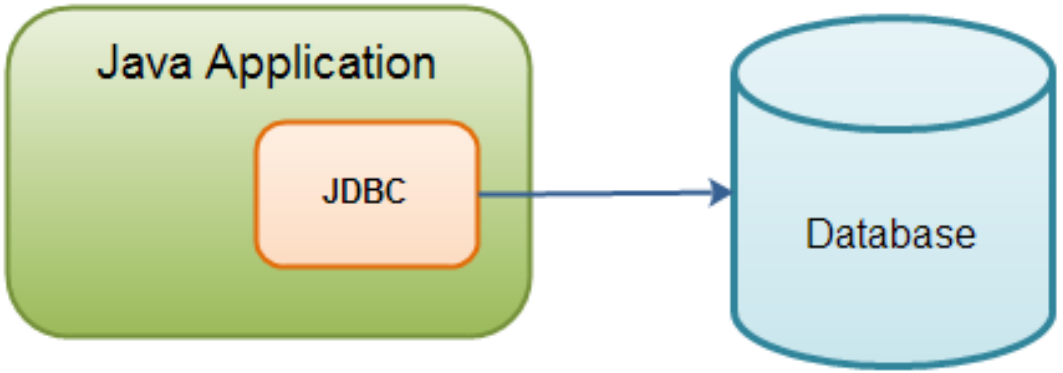
- Implements a synchronized hash table.
- Thread-safe but with additional overhead.
- Does not allow null keys or values.
- Generally slower than HashMap due to synchronization.
- **When to use**: When a thread-safe implementation of a map is required.

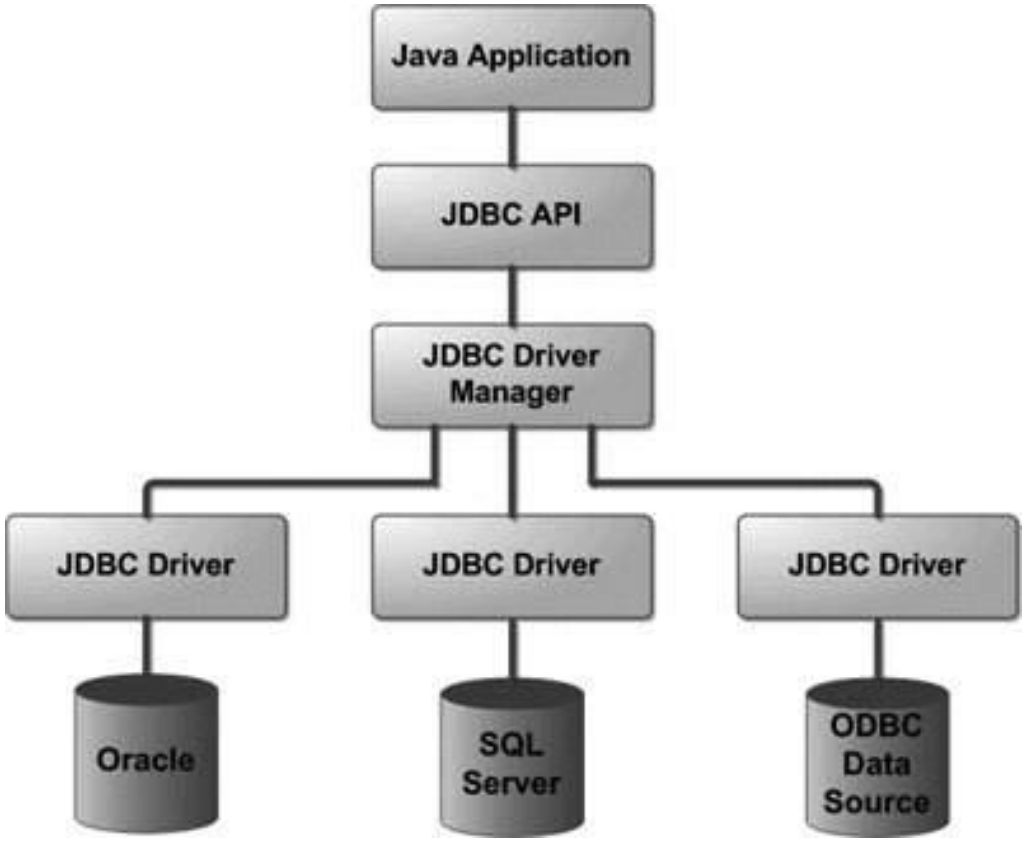
Feature	HashMap	LinkedHashMap	TreeMap	Hashtable
Underlying Data Structure	Hash table	Hash table + Linked list	Red-Black tree	Hash table
Iteration Order	Unordered	Insertion order	Sorted order	Unordered
Allow null Keys/Values	Yes	Yes	No (keys), Yes (values)	No
Performance (Add, Remove, Contains)	O(1)	O(1) (slightly slower than HashMap)	O(log n)	O(1) (synchronized, slower)
Thread-Safety	No	No	No	Yes
Memory Overhead	Less	More (due to linked list)	More (due to tree structure)	Less

PRODUCT INVENTORY SYSTEM

- Create a Product class with the following attributes:
  - productId (int)
  - name (String)
  - price (double)
- Create an Inventory class that contains an List of Product objects. Implement the following methods
  - A constructor that initializes the ArrayList of products.
  - addProduct(Product product) method that adds a product to the inventory.
  - displayInventory() method that prints the details of all products in the inventory
  - removeProduct(id) method remove product from an array.
  - searchProduct(name) method search product from an array by name.
  - Allow user to exit (system.exit())
  - If Needed handle Exceptions using try catch.

JDBC


Java Application
JDBC
Database


Java Application
JDBC API
JDBC Driver Manager
JDBC Driver
JDBC Driver
JDBC Driver
Oracle
SQL Server
ODBC Data Source

(Type-1) JDBC-ODBC Bridge driver
(Type-2) Native-API-Partly-Java driver
(Type-3) Net-All-Java driver
(Type-4) Native-Protocol-All-Java-Driver

Property	Type-1	Type-2	Type-3	Type-4
Conversion	From JDBC calls to ODBC calls	From JDBC calls to native library calls	From JDBC calls to middle-wear specific calls	From JDBC calls to Data Base specific calls
Implemented-in	Only java	Java + Native language	Only java	Only java
Architecture	Follow 2-tier architecture	Follow 2-tier architecture	Follow 3-tier architecture	Follow 2-tier architecture
Platform-independent	NO	NO	YES	YES
Data Base independent	YES	NO	YES	NO
Thin or Thick	Thick	Thick	Thick	Thin

LET'S CONNECT WITH DB

```
public class ConnectDemo {

    public static void main(String[] args) {

        String driver="com.mysql.cj.jdbc.Driver";
        String url="jdbc:mysql://localhost:3306/techABC";
        String username="root";
        String password="Sonam@123";
        //load driver
        try {
            Class.forName(driver);
            Connection conn=DriverManager.getConnection(url,username,password);
            if(conn!=null) {
                System.out.println("Connected");
            }else {
                System.out.println("Not Connected with DB");
            }
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        }catch(SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Load driver
Connect with DB

STATEMENT

- The Statement interface is used to execute SQL queries and update statements against a database.
- There are three main types of statements in JDBC:
  - **Statement**: Used for executing simple SQL queries without parameters.
  - **PreparedStatement**: Used for executing precompiled SQL queries with parameters.
  - **CallableStatement**: Used for executing stored procedures.

STATEMENT

- The Statement interface is used for executing a static SQL statement and returning the results it produces.
- It is best suited for executing DDL (Data Definition Language) and simple SQL queries.

```
//load driver
try {
    Class.forName(driver);
    Connection conn=DriverManager.getConnection(url,username,password);
    //Static Insertion
    Statement statement= conn.createStatement();

    String sql= "insert into employee (id,name,email,age,phone) values"
            + "(1,'Alex','alex@gmail.com',34,9876543210)";

    int i=statement.executeUpdate(sql);
    System.out.println(i+" Records Inserted in DB");
```

PREPARED STATEMENT

- The PreparedStatement interface is used to execute parameterized SQL queries. It is precompiled and allows for setting input parameters at runtime.
- This is more efficient and secure than using a Statement for executing queries with parameters.

```
Connection conn = DBConfig.getConnection();
String sql = "insert into employee (id,name,email,age,phone) values(?,?,?,?,?)";
try {
    PreparedStatement stmt = conn.prepareStatement(sql);
    stmt.setInt(1, id);
    stmt.setString(2, name);
    stmt.setString(3, email);
    stmt.setInt(4, age);
    stmt.setBigDecimal(5, new BigDecimal(phone));

    int i= stmt.executeUpdate();
    System.out.println(i+" Records inserted in DB");
} catch (SQLException e) {
    e.printStackTrace();
}
```

CALLABLE STATEMENT

- The CallableStatement interface is used to execute stored procedures in a database.
- It allows calling database stored procedures with both input and output parameters.

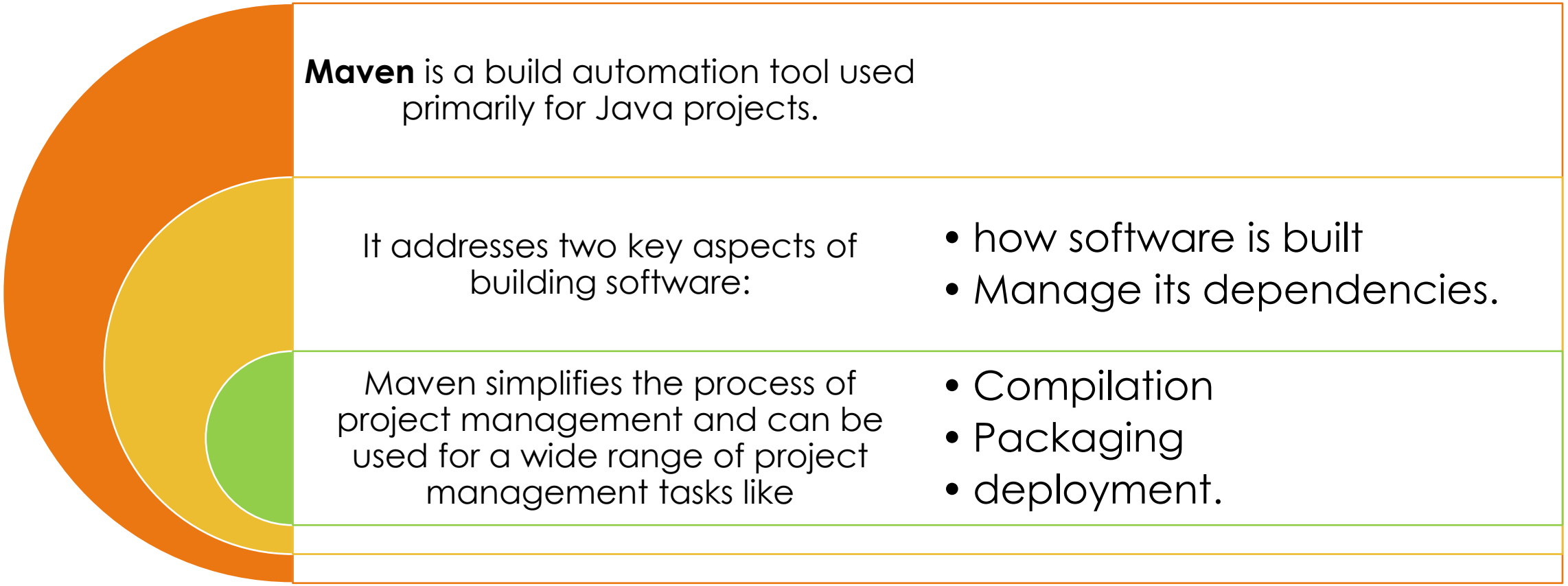
LET'S WORK ON USE CASE USING DAO PATTERN

- Create util class to keep driver, url, username, password
- Create config class
- Create model
- Create dao interface
- Create daoImpl implementation
- Use it in main class

IMPLEMENT PROJECT USECASE

- Create model product
- Create interface ProductDao
- Implement all its CRUD methods using ProductDaoImpl
- Use it as per requirements.

MAVEN


Maven is a build automation tool used primarily for Java projects.
It addresses two key aspects of building software:
• how software is built
• Manage its dependencies.
Maven simplifies the process of project management and can be used for a wide range of project management tasks like
• Compilation
• Packaging
• deployment.

KEY CONCEPTS OF MAVEN

Project Object Model (POM):

- It is core unit of work in Maven.
- This is an XML file that contains information about the project and configuration details used by Maven to build the project.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
    <modelVersion>4.0.0</modelVersion>
    <groupId>org.example</groupId>
    <artifactId>TestCase</artifactId>
    <version>1.0-SNAPSHOT</version>
    <properties>
        <maven.compiler.source>17</maven.compiler.source>
        <maven.compiler.target>17</maven.compiler.target>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    </properties>
</project>
```

KEY CONCEPTS OF MAVEN

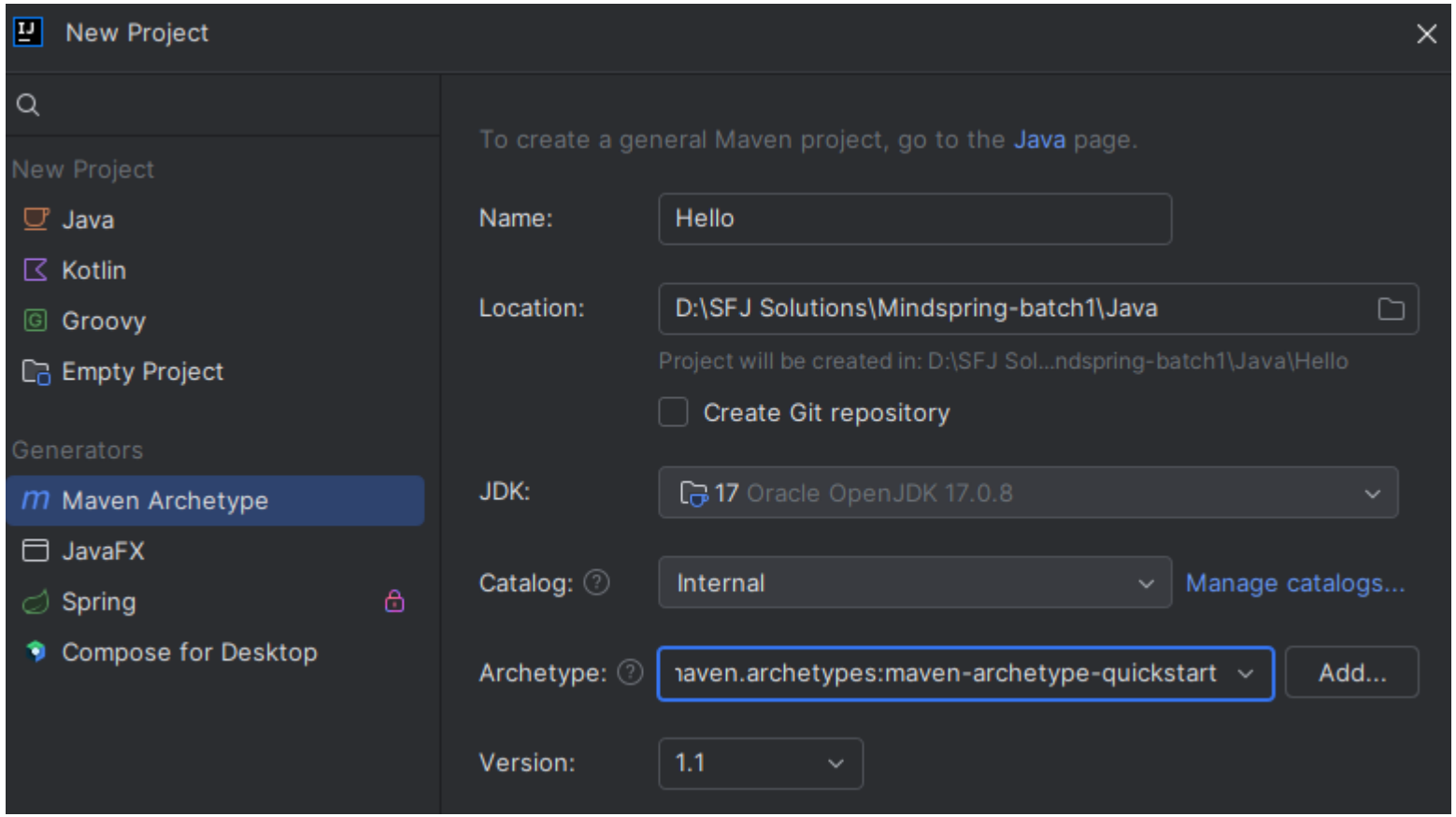
Dependencies:

- Dependencies are libraries or JAR files that your project needs to compile, test, and run.
- They are specified in the pom.xml file.

```
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-api</artifactId>
        <version>5.10.3</version>
        <scope>test</scope>
    </dependency>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>5.10.3</version>
        <scope>test</scope>
    </dependency>
</dependencies>
```

LET'S CREATE MAVEN PROJECT

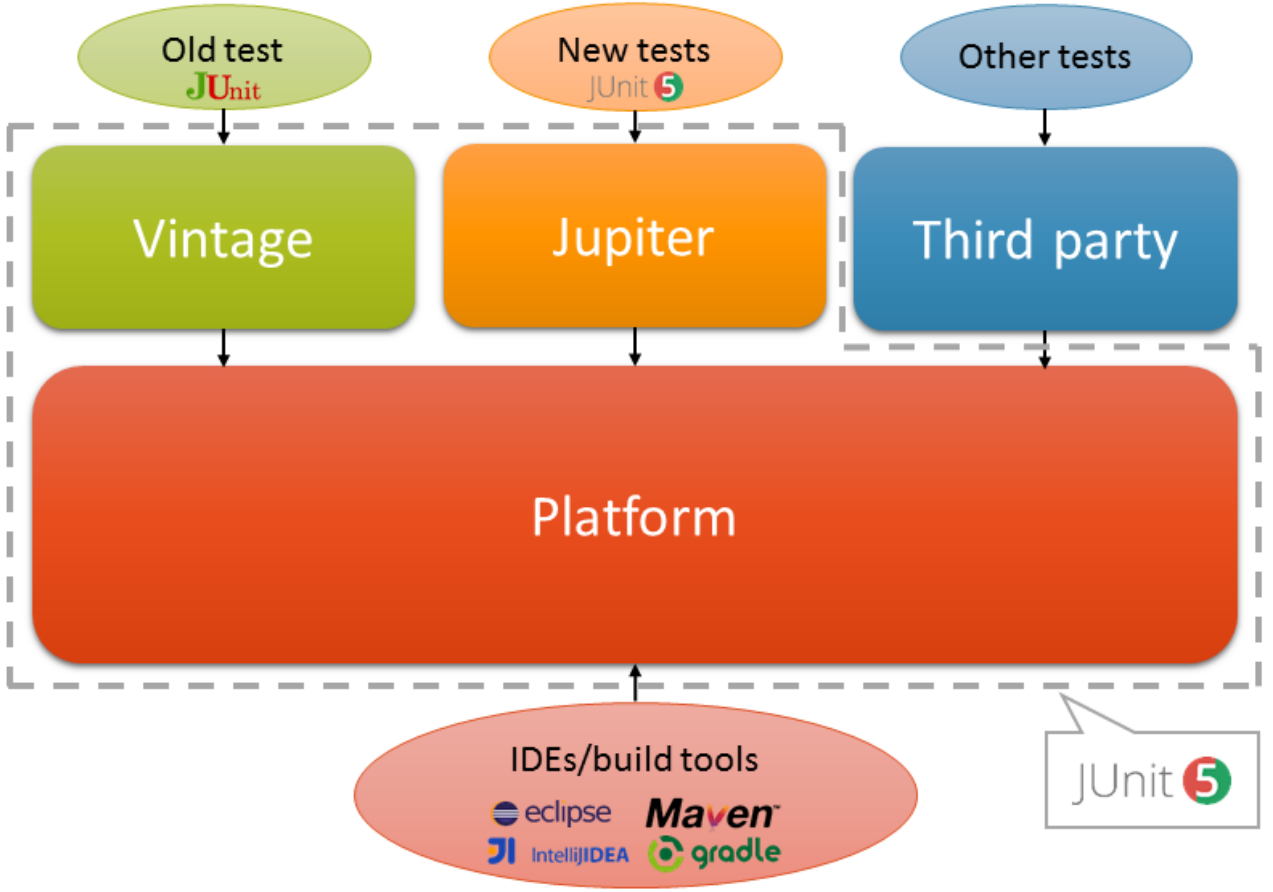
- Select New Project
- Select Maven
- It will ask you to select archetype
  - Java project: quick start
  - Web project: webapp
- Click on create



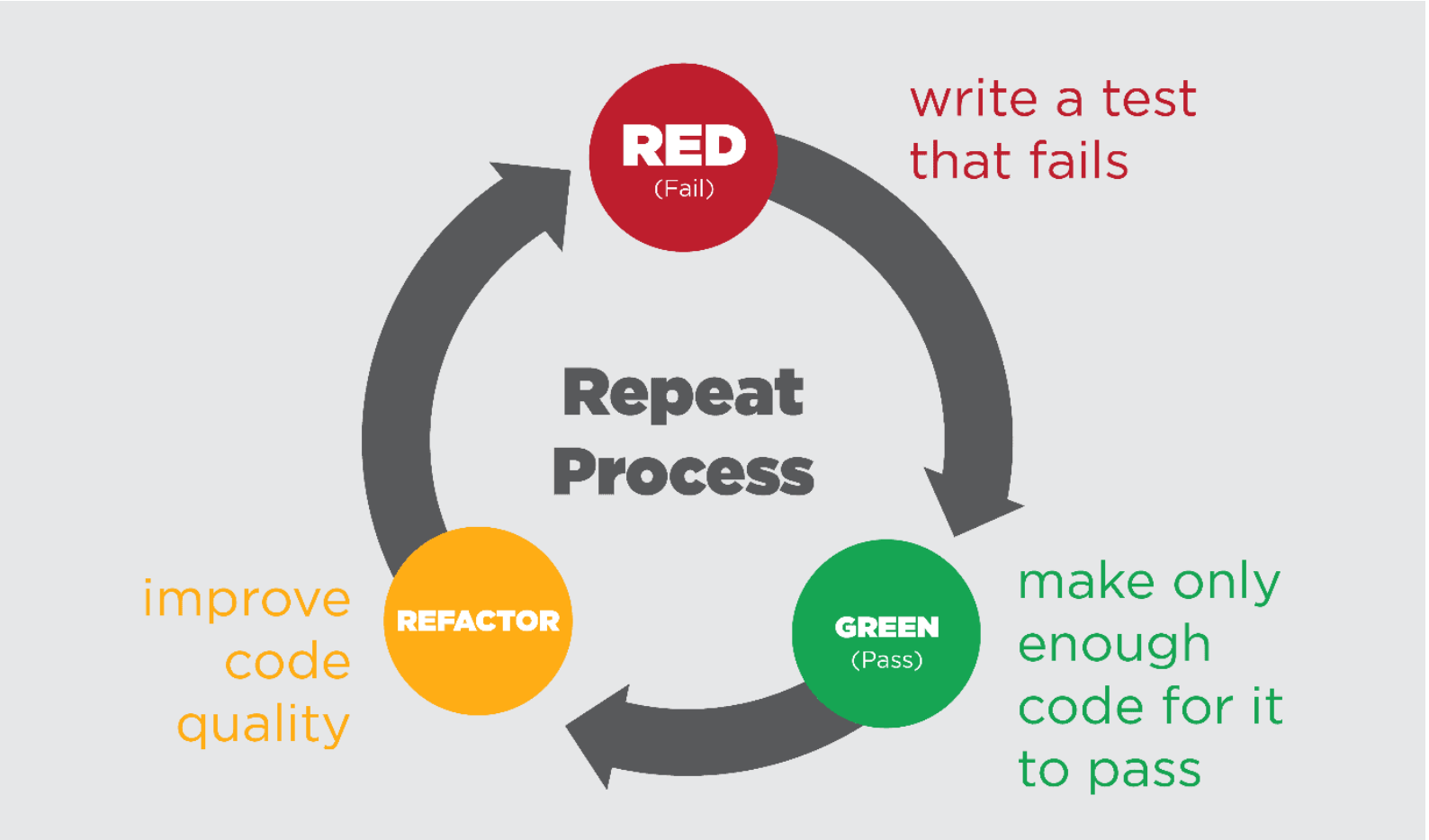
JUNIT

Unit testing is a software testing methodology where individual units or components of a software system are tested independently to ensure that each unit works as expected.

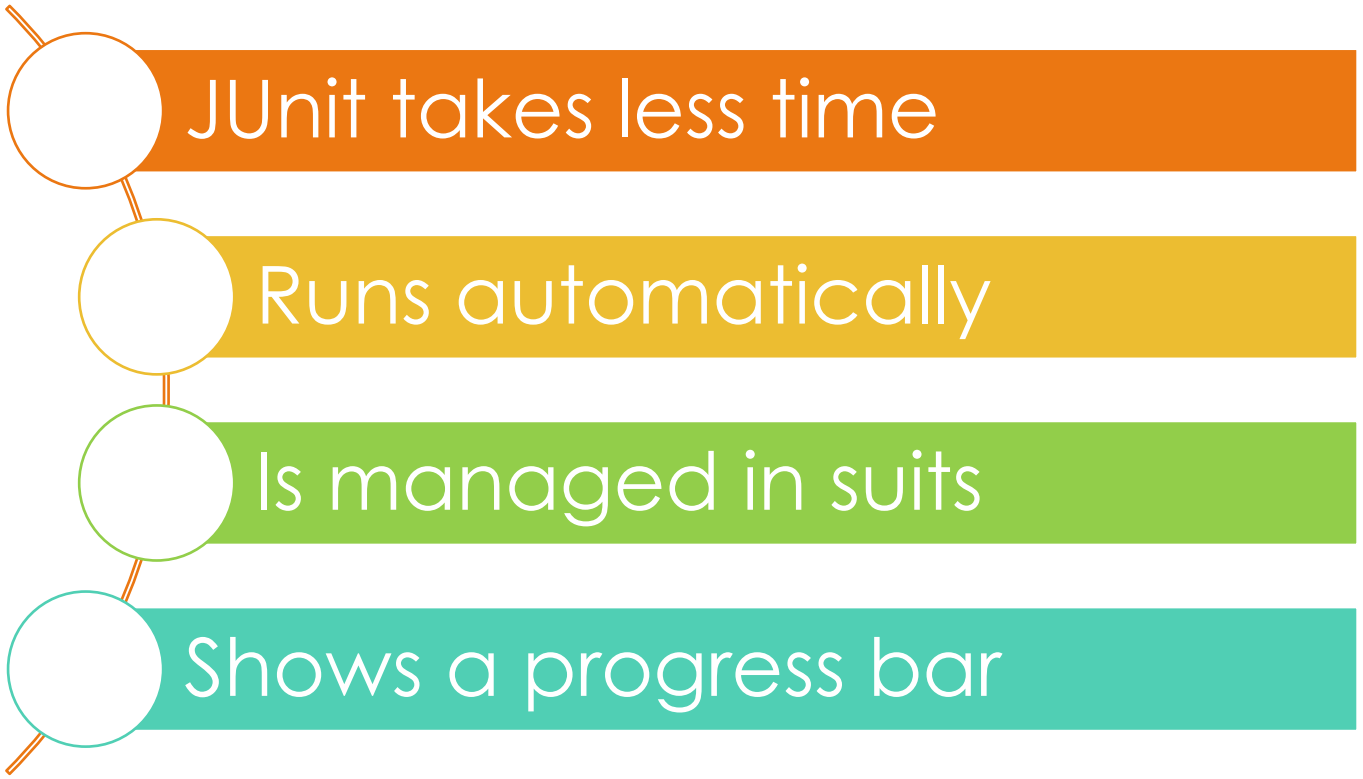
JUnit plays an important role in Unit Testing as it provides a framework for writing and executing automated tests, ensuring that code changes do not break the existing functionality.



TDD APPROACH



FEATURES OF JUNIT



LET'S SETUP MAVEN PROJECT

- Create one Maven project
- Add 2 dependency for Junit:
  - Junit jupiter
  - Junit engine
- Once its included create once file named calculator in src
- Test case inside test folder
- Run the testcase

ANNOTATION IN JUNIT

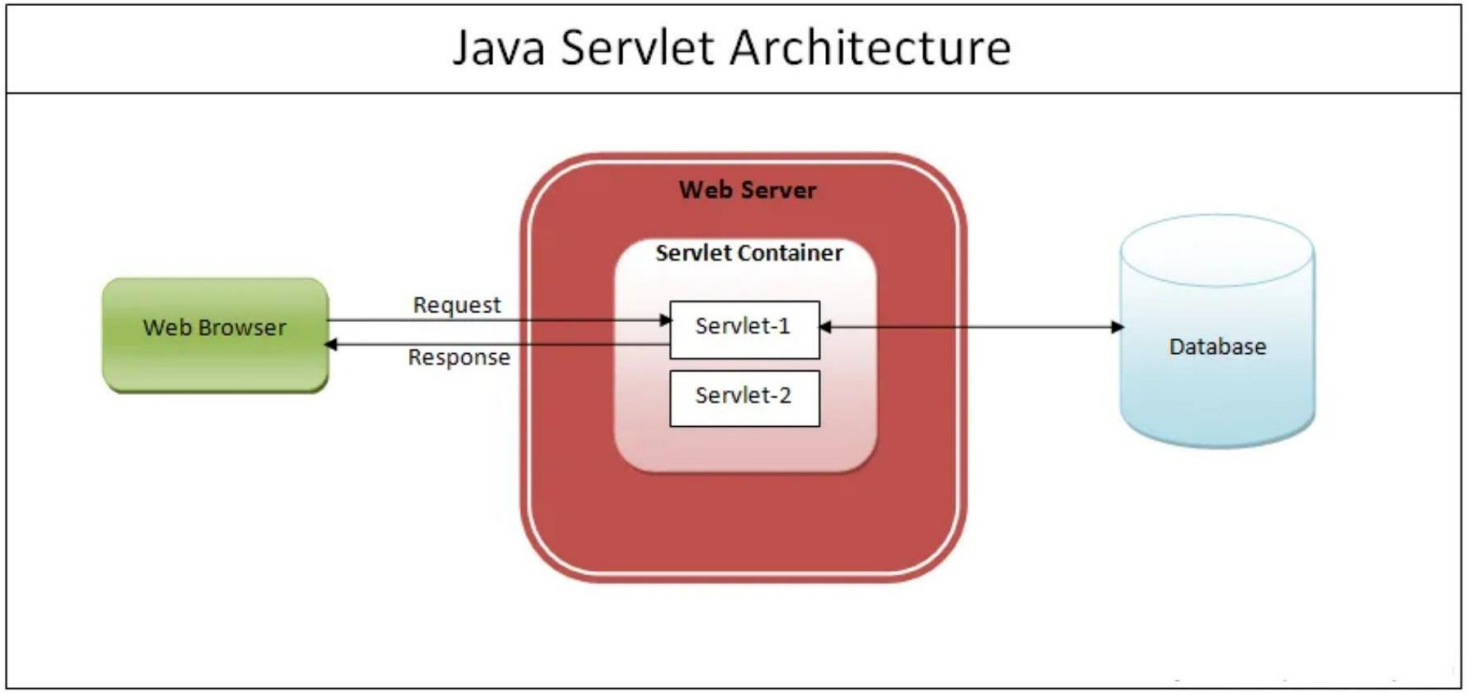
@Test	• It provides information that the public void method it is attached to can be executed as a test case.
@BeforeEach	• It helps a method to be executed before every test method.
@AfterEach	• It helps a method to be executed after the test method.
@Ignore	• It is used to ignore a test that will not be executed.

WRITE JUNIT TEST CASE FOR INVENTORY

- Write test case for addProduct
- Write test case for removeProduct
- Write test case for getAllProducts
- Write test case for searchProduct
- Write test case for getProductById

INTRO TO SERVLET

- Java Servlets are the Java programs that run on the Java-enabled web server or application server.
- They are used to handle the request obtained from the web server, process the request, produce the response, and then send a response back to the web server.



INTRODUCTION TO

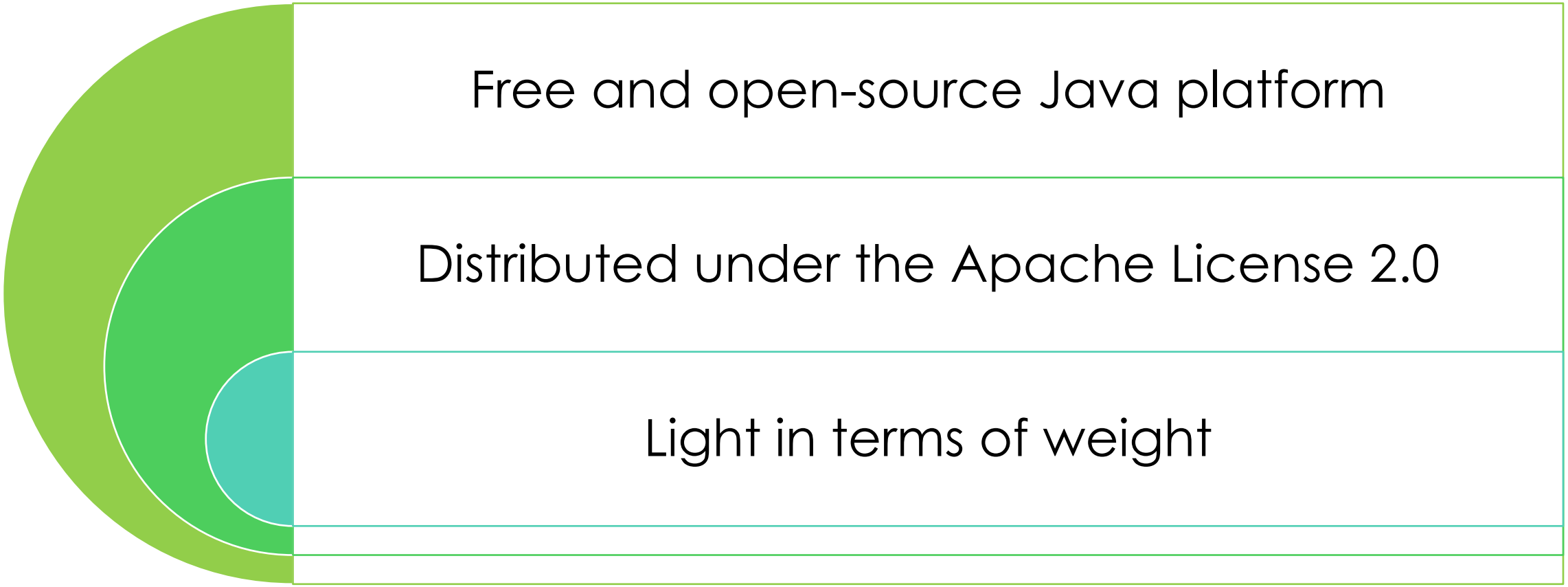
INTRO TO SPRING

Spring is a powerful Java corporate application development framework.

Spring is a challenging framework for writing high-performance, testable, and reusable code.

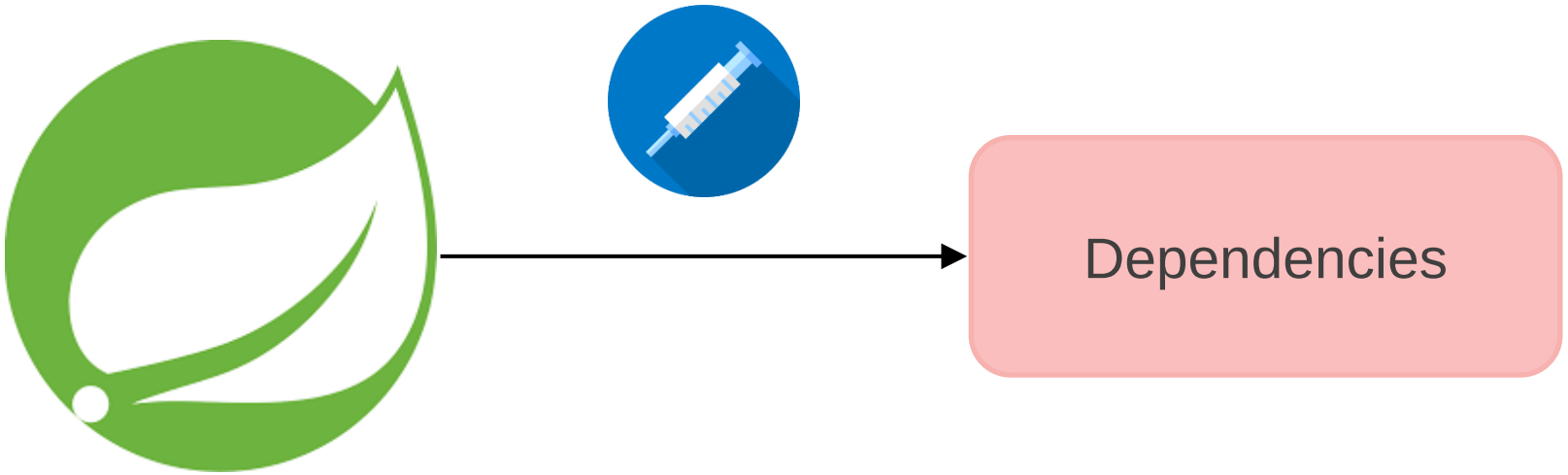
Spring provides a comprehensive programming and configuration model for modern Java-based applications, enabling developers to build scalable, modular, and easy-to-maintain applications with less boilerplate code.

WHY SPRING?


Free and open-source Java platform
Distributed under the Apache License 2.0
Light in terms of weight

DEPENDENCY INJECTION

- Dependency Injection is the basic functionality given by Spring IoC.
- Spring-Core is responsible for injecting dependencies.
- It injects the dependencies with the help of
  - Setter Methods
  - Constructor



DEPENDENCY INJECTION

- The design principles of IoC
- Emphasize keeping the Java classes independent from each other
- Get free by container from creating an object and maintenance

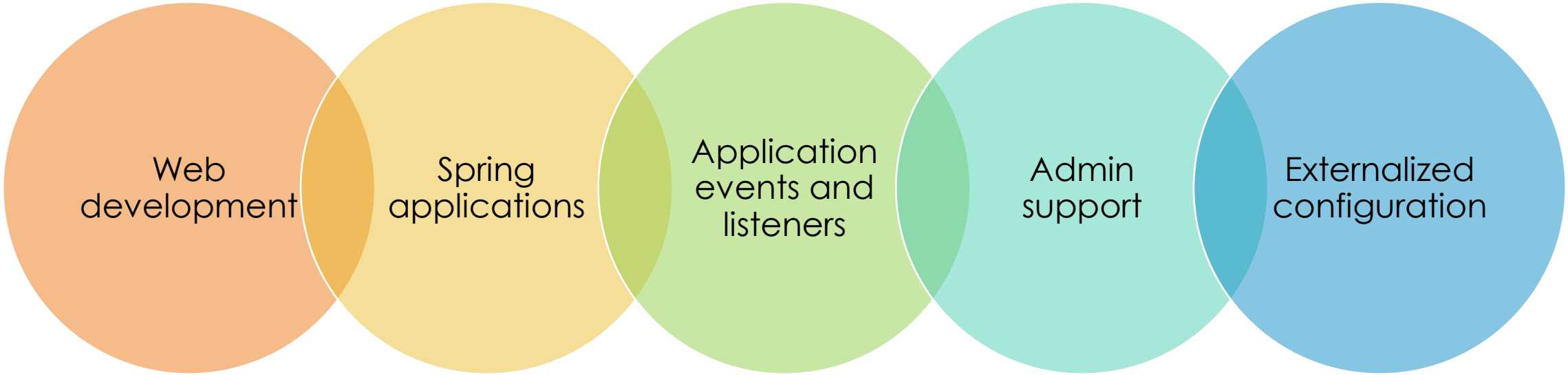
ACTIVITY

- Let’s configure Spring Project
- Create Maven project
- Add dependency:
  - Spring-core
  - Spring-context
  - spring-beans
- Create configuration file in resources in beans.xml
- Read object using ApplicationContext

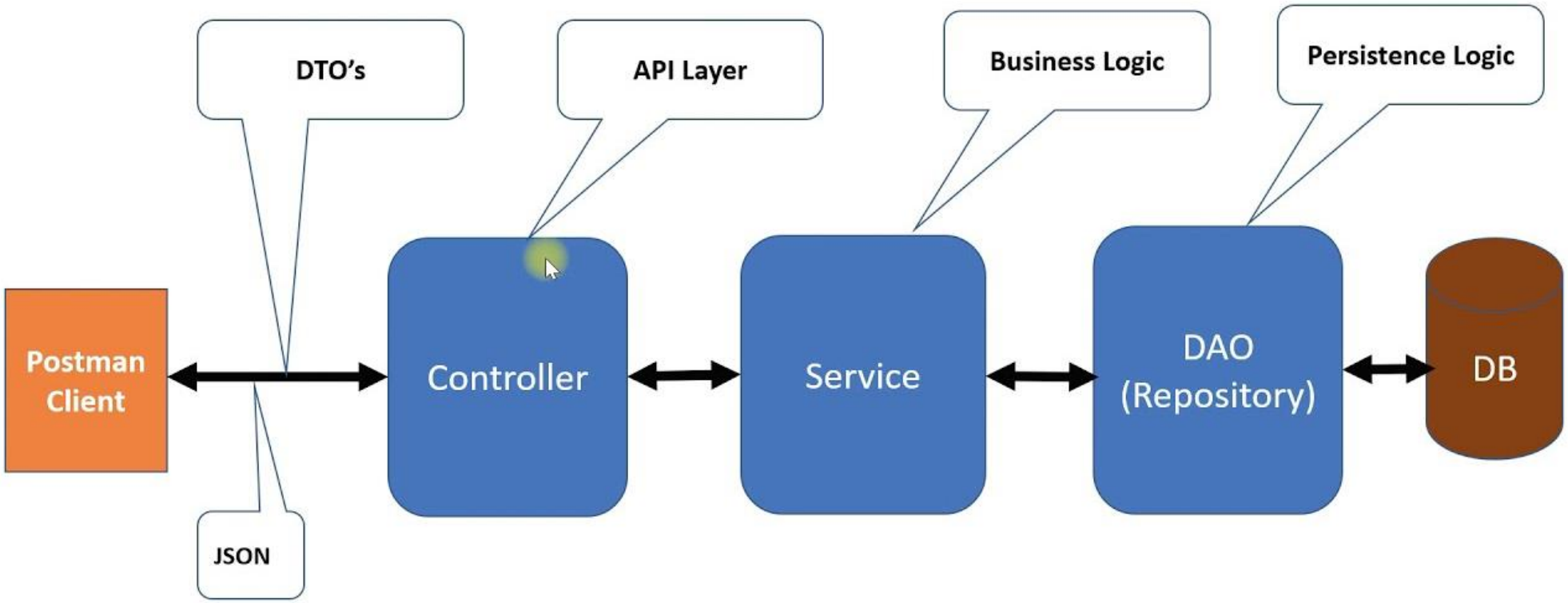
INTRODUCTION TO SPRINGBOOT

- It refers to a project built on top of the Spring framework.
- It facilitates easy and fast configuration and helps run simple and web-based applications.
- It is a part of the Spring module that gives the RAD (Rapid Application Development) feature to the spring framework.
- It creates a standalone Spring-based application.
- It is a combination of the Spring framework and the embedded servers.

INTRODUCTION TO SPRINGBOOT

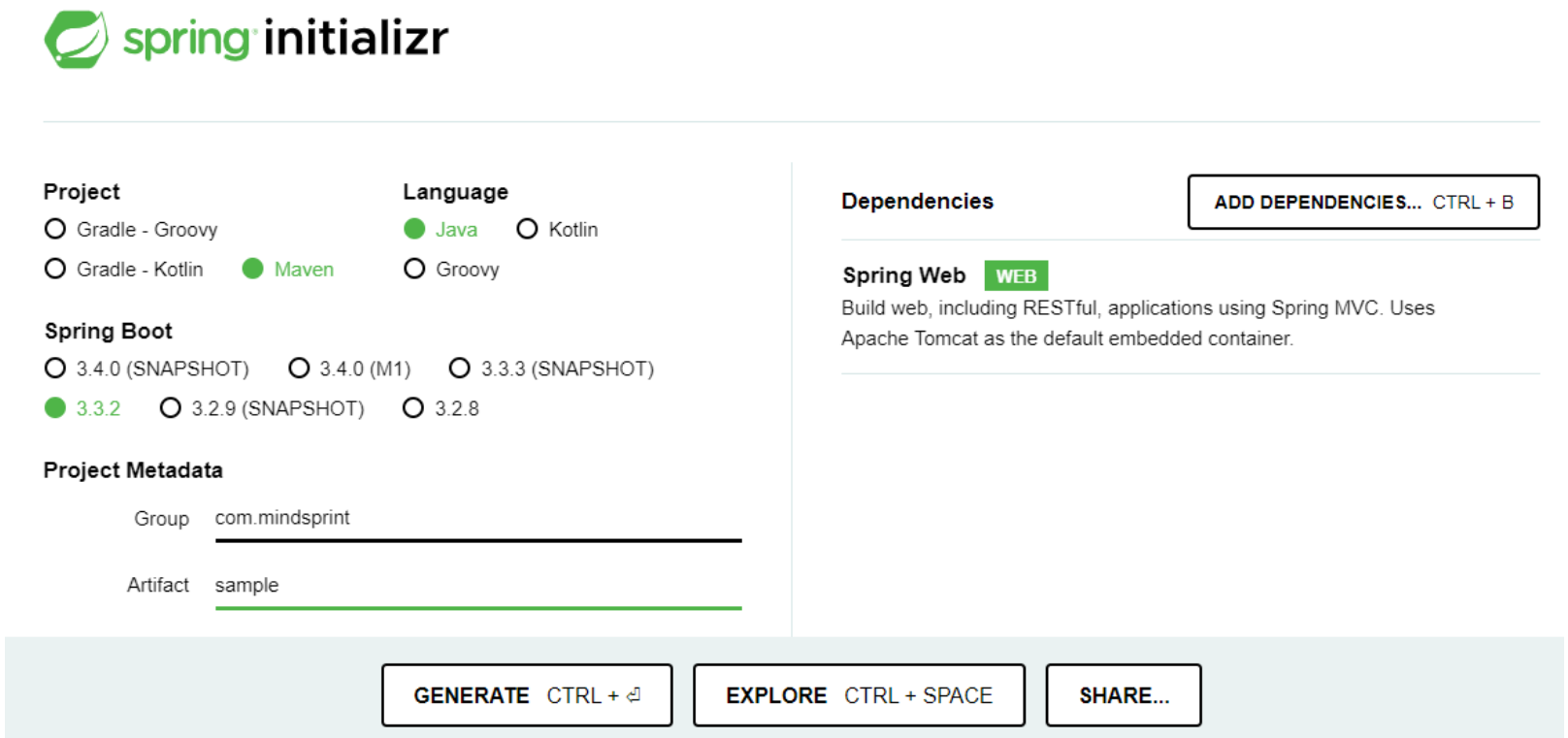


SPRING BOOT ARCHITECTURE



LET'S CREATE SPRING BOOT PROJECT

- Go to the spring initializer website https://start.spring.io/



- Import project by using open in intellJ
- Once its is import create one controller
- Run the project and check: localhost:8080/api/name

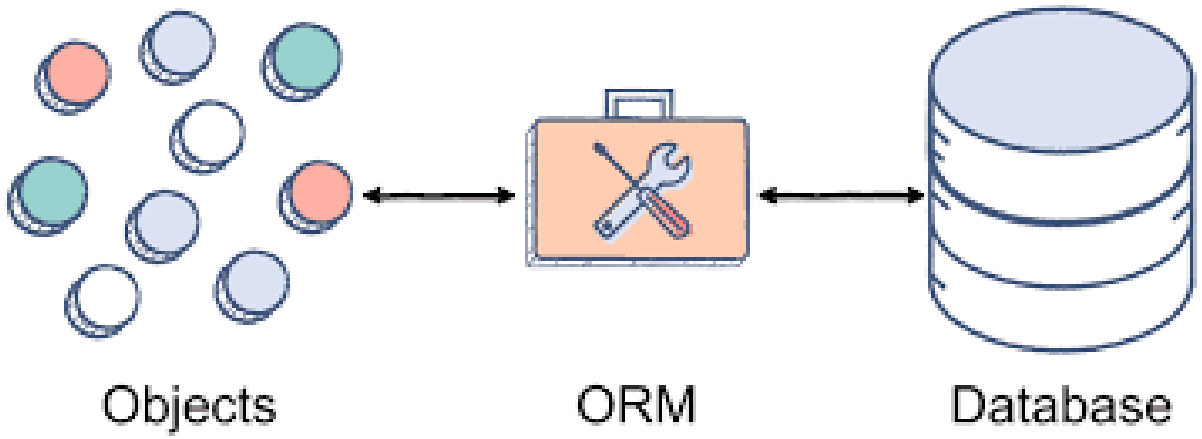
```
@RestController  no usages  new *
@RequestMapping("/api/name")
public class Controller {
    @GetMapping  no usages  new *
    public String getMyData(){
        return  "Sonam Soni";
    }
}
```

REST API USING SPRINGBOOT

- Activity

INTRO TO ORM

- **Object-Relational Mapping (ORM)** is a programming technique used to convert data between incompatible type systems in object-oriented programming languages.
- It allows developers to interact with a database using the concepts of objects and classes in their preferred programming language instead of SQL.



ORM FEATURES

Abstraction:

- ORM abstracts the database interactions, allowing developers to focus on business logic rather than database-specific code.

Mapping:

- It maps database tables to classes, table rows to objects, and table columns to object attributes.

Automatic SQL Generation:

- ORM frameworks automatically generate SQL queries based on the operations performed on the objects.

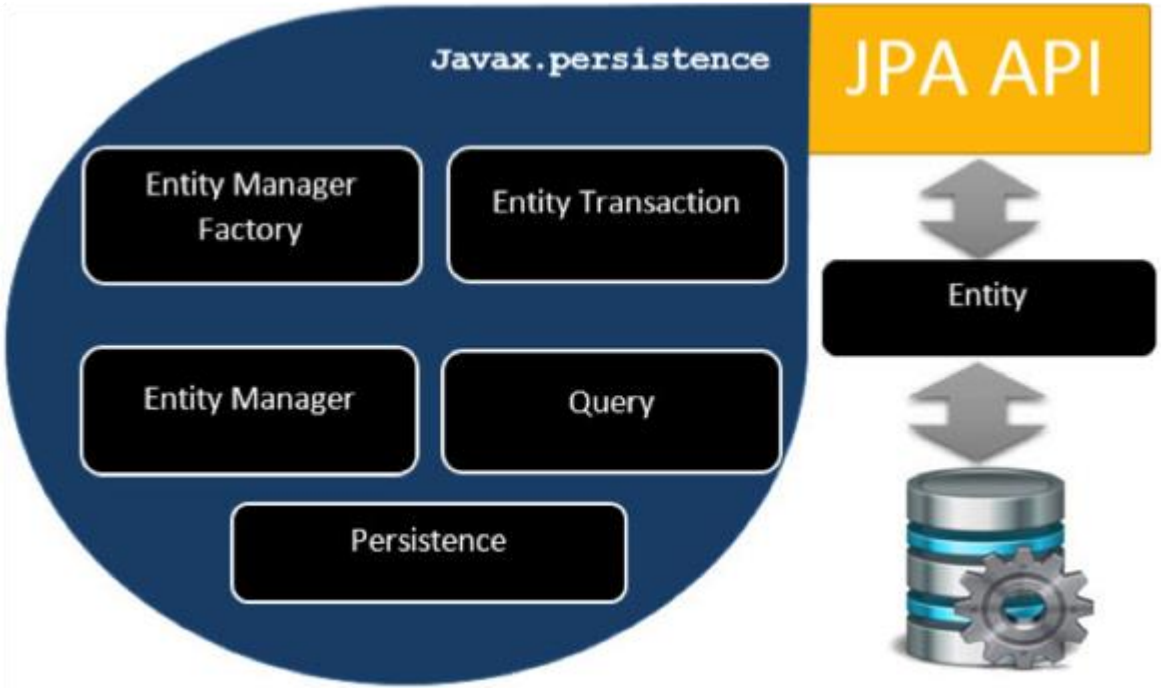
Object-Oriented:

- Developers can use object-oriented principles, such as inheritance and polymorphism, in their data models.

JAVA PERSISTENCE API (JPA)

Java Persistence API (JPA) is a specification for ORM in Java. It defines a set of rules and guidelines for managing relational data in Java applications.

JPA is not a framework by itself but provides the necessary guidelines that ORM frameworks (like Hibernate) can implement.



ENTITY

- An entity represents a table in a relational database.
- Each instance of an entity corresponds to a row in the table.

```
@Entity  5 usages  new *
public class User {
    @Id  2 usages
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;  2 usages
    private String email;  2 usages

    public Long getId() {  1 usage  new *
        return id;
    }
```

LET'S IMPLEMENT JPA WITH SPRINGBOOT

- Activity

PROPERTIES FILE

```
spring.datasource.url=jdbc:mysql://localhost:3306/restapi
spring.datasource.username=root
spring.datasource.password=Sonam@123
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MySQLDialect
spring.jpa.show-sql=true
```

ONE TO MANY RELATIONSHIPS

```
@Entity
public class Author {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String name;

    @OneToMany(mappedBy = "author", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private List<Book> books;
```

IN THE OTHER CLASS MANY TO ONE

```
@Entity
public class Book {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "author_id")
    private Author author;
```

HOW TO INJECT DATA

- Create Book:

```
{
   "title": "New Book",
   "author": {
      "id": 1
   }
}
```

- Create Author:

```
{
   "name": "Author Name",
   "books": [
      {
         "title": "Book 1"
      },
      {
         "title": "Book 2"
      }
   ]
}
```

MANY TO MANY RELATIONSHIPS

- create two entities, Student and Course, which have a many-to-many relationship.

```
@Entity
public class Student {

    @Id
    @GeneratedValue(strategy =
GenerationType.IDENTITY)
    private Long id;

    private String name;

    @ManyToMany
    @JoinTable(
      name = "student_course",
      joinColumns = @JoinColumn(name =
"student_id"),
      inverseJoinColumns =
@JoinColumn(name = "course_id"))
    private Set<Course> courses = new
HashSet<>();
```

COURSE CLASS

```
@Entity
public class Course {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;

    private String title;

    @ManyToMany(mappedBy = "courses")
    private Set<Student> students = new HashSet<>();
```