**Advance Java**

**Dt. 10th July 2024**

**package com.wipro.service;**

**/\*\***

**\* Runnable is a Functional Interface, an interface with only one abstract method**

**\*/**

**public class MyRunnable implements Runnable{**

**@Override**

**public void run() {**

**System.*out*.println(Thread.*currentThread*());**

**System.*out*.println("Working on a job..");**

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**System.*out*.println("Job done");**

**}**

**}**

**package com.wipro.app;**

**import com.wipro.service.MyRunnable;**

**/\*\***

**\* It common to have instance of implementation class pointed by its**

**\* interface type.**

**\* Similarly it is common to have instance of a sub-class pointed by its super class.**

**\***

**\* The above procedure is required to achieve polymorphic behaviour.**

**\***

**\*/**

**public class MyRunnableDemo {**

**public static void main(String[] args) {**

**System.*out*.println(Thread.*currentThread*());**

**// Create a Runnable object**

**Runnable runnable = new MyRunnable();**

**//Attach Runnable object to Thread object**

**// Thread thread1 = new Thread(runnable);**

**Thread thread1 = new Thread(runnable,"worker-1");**

**thread1.start();//implicitly invokes run() method**

**try {**

**/\***

**\* thread1 says to main thread to join me. i.e. main thread**

**\* goes to non-runnable state until worker thread has completed its task**

**\* and then will come back to runnable state**

**\*/**

**thread1.join();**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**Thread thread2 = new Thread(runnable, "worker-2");**

**thread2.start();**

**try {**

**thread2.join();**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**System.*out*.println("End of main() method");**

**}**

**}**

**—--------------------------------------------------------------------------------------------------**

**package com.wipro.service;**

**public class MyThread extends Thread{**

**public MyThread() {**

**super();**

**}**

**public MyThread(String name) {**

**super(name); // invoking 1-arg constructor of Thread class**

**}**

**@Override**

**public void run() {**

**super.run();**

**System.*out*.println(Thread.*currentThread*());**

**System.*out*.println("processing..");**

**try {**

**Thread.*sleep*(1000);**

**} catch (InterruptedException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**System.*out*.println("End of worker thread");**

**}**

**}**

**package com.wipro.app;**

**import com.wipro.service.MyThread;**

**public class MyThreadDemo {**

**public static void main(String[] args) {**

**System.*out*.println(Thread.*currentThread*());**

**Thread thread1 = new MyThread("worker-1");**

**thread1.start();//implicitly invokes run() method**

**try {**

**thread1.join();**

**} catch (InterruptedException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**System.*out*.println("End of main thread");**

**}**

**}**

**—-------------------------------------------------------------------------------------------**

**Race Condition**

When two or more threads are competing to gain access to the same shared data is called **race condition**.

Race conditions may lead to data inconsistencies because while one thread is reading other thread is writing to shared data.

The section of the code that leads to race conditions is called the **critical section**.

**How to avoid race conditions?**

By proper synchronization of the code, we can avoid race conditions.

There are two ways:

1. **Synchronized method**: When a method is synchronized, only one thread can enter into the method at a time.

public **synchronized** returnType methodName(){

}

1. **Synchronized block**: Instead of restricting the threads from entering into the method, block only the critical section so that only one thread at a time can enter into the block.

public returnType methodName(){

—----------------;

**synchronized(shared data){**

**—----------------;**

**}**

*—-------------------;*

}

**package** com.wipro.service;

/\*\*

\* StringBuffer methods are synchronized by default hence the StringBuffer objects

\* are thread-safe.

\*

\* StringBuilder methods are not synchronized hence not thread-safe.

\*

\* Note: Both are mutable objects

\*/

**public** **class** SharedObject **implements** Runnable{

**private** **static** StringBuilder *sharedData*= **new** StringBuilder("Welcome");

@Override

**public** **void** run() {

**int** length = *sharedData*.length();

**for**(**int** i=0;i<length;i++) {

System.***out***.println(Thread.*currentThread*().getName() + "-"+ *sharedData*.append("x"));

}

}

}

**package** com.wipro.app;

**import** com.wipro.service.SharedObject;

**public** **class** SyncDemo {

**public** **static** **void** main(String[] args) {

Runnable runnable = **new** SharedObject();

Thread thread1 = **new** Thread(runnable,"worker-1");

Thread thread2 = **new** Thread(runnable,"worker-2");

thread1.start();

thread2.start();

}

}

**package com.wipro.service;**

**/\*\***

**\* StringBuffer methods are synchronized by default hence the StringBuffer objects**

**\* are thread-safe.**

**\***

**\* StringBuilder methods are not synchronized hence not thread-safe.**

**\***

**\* Note: Both are mutable objects**

**\*/**

**public class SharedObject implements Runnable{**

**private static StringBuilder *sharedData*= new StringBuilder("Welcome");**

**// @Override**

**// public synchronized void run() {**

**//**

**// int length = sharedData.length();**

**// for(int i=0;i<length;i++) {**

**// System.out.println(Thread.currentThread().getName() + "-"+ sharedData.append("x"));**

**// }**

**//**

**// }**

**@Override**

**public void run() {**

**int length = *sharedData*.length();**

**synchronized(*sharedData*) {**

**for(int i=0;i<length;i++) {**

**System.*out*.println(Thread.*currentThread*().getName() + "-"+ *sharedData*.append("x"));**

**}**

**}**

**}**

**}**

**Anonymous Inner Class**

**A class defined inside a method is called an inner class. An inner class which has no name is called anonymous inner class.**

**public class Demo{**

**public static void main(String [] args){**

**//inner class**

**class A{**

**—-----p**

**}**

**}**

**}**

**Syntax of anonymous inner class:**

**(new SuperClassName() {**

**public void method(){**

**}**

**}).method();**

**—-------------------------------------------------------**

**package com.wipro.model;**

**//POJO : Plain Old Java Object**

**public class Person implements Comparable<Person>{**

**private Long adharCard;**

**private String name;**

**private Long mobile;**

**public Person() {**

**}**

**public Person(Long adharCard, String name, Long mobile) {**

**super();**

**this.adharCard = adharCard;**

**this.name = name;**

**this.mobile = mobile;**

**}**

**public Long getAdharCard() {**

**return adharCard;**

**}**

**public void setAdharCard(Long adharCard) {**

**this.adharCard = adharCard;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public Long getMobile() {**

**return mobile;**

**}**

**public void setMobile(Long mobile) {**

**this.mobile = mobile;**

**}**

**@Override**

**public String toString() {**

**return "Person [adharCard=" + adharCard + ", name=" + name + ", mobile=" + mobile + "]";**

**}**

**//p1.compareTo(p2)**

**@Override**

**public int compareTo(Person o) {**

**return this.getAdharCard().compareTo(o.getAdharCard());**

**}**

**}**

**—--------------------------------------------------------------**

**package com.wipro.app;**

**import java.util.Arrays;**

**import com.wipro.model.Person;**

**/\***

**\* The current values in the instance variables of an object is called object state.**

**\*/**

**public class Demo {**

**public static void main(String[] args) {**

**//Anonymous inner class**

**(new Object(){**

**public void greeting() {**

**System.*out*.println("Welcome to anonymous inner class");**

**}**

**}).greeting();**

**//com.wipro.model.Person@63961c42**

**//63961c42 is hexa-decimal representation of the object's hashCode**

**Person person = new Person(123456783465L,"Smith",9876879879L);**

**/\* To display object state rather than reference, we need to**

**\* override toString() method which returns String representation**

**\* of the object**

**\*/**

**System.*out*.println(person);**

**System.*out*.println(person.hashCode()); //1670782018**

**Person [] persons= new Person[] {**

**new Person(423456783465L,"Smith",9876879879L),**

**new Person(523456783495L,"Clarke",7876879855L),**

**new Person(323456783405L,"Jones",8876879870L)**

**};**

**System.*out*.println("After sorting...");**

**Arrays.*sort*(persons);**

**System.*out*.println(Arrays.*toString*(persons));**

**}**

**}**

**Note:**

**When objects of user-defined type have to be compared based on key attribute, then the class has to implement Comparable interface and override compareTo() method.**

**When objects of user-defined type have to be compared based on non-key attribute, then create a new class that implements Comparator interface and override compare() method or instead of creating a new class, go for anonymous inner class(which can later be replaced with Lamda expressions)**

**—---------------------------------------------------------------**

**package com.wipro.app;**

**import java.util.Arrays;**

**import java.util.Comparator;**

**import com.wipro.model.Person;**

**/\***

**\* The current values in the instance variables of an object is called object state.**

**\*/**

**public class Demo {**

**public static void main(String[] args) {**

**//Anonymous inner class**

**(new Object(){**

**public void greeting() {**

**System.*out*.println("Welcome to anonymous inner class");**

**}**

**}).greeting();**

**//com.wipro.model.Person@63961c42**

**//63961c42 is hexa-decimal representation of the object's hashCode**

**Person person = new Person(123456783465L,"Smith",9876879879L);**

**/\* To display object state rather than reference, we need to**

**\* override toString() method which returns String representation**

**\* of the object**

**\*/**

**System.*out*.println(person);**

**System.*out*.println(person.hashCode()); //1670782018**

**Person [] persons= new Person[] {**

**new Person(423456783465L,"Smith",9876879879L),**

**new Person(523456783495L,"Clarke",7876879855L),**

**new Person(323456783405L,"Jones",8876879870L)**

**};**

**System.*out*.println("After sorting based on adhar cards...");**

**Arrays.*sort*(persons);**

**System.*out*.println(Arrays.*toString*(persons));**

**System.*out*.println("After sorting based on person names...");**

**/\***

**\* the second argument is an anonymous inner class that**

**\* implements Comparator interface**

**\*/**

**Arrays.*sort*(persons, new Comparator<Person>() {**

**@Override**

**public int compare(Person o1, Person o2) {**

**return o1.getName().compareTo(o2.getName());**

**}**

**});**

**System.*out*.println(Arrays.*toString*(persons));**

**}**

**}**

**Concurrency**

The set of interfaces and classes related to concurrency belong to a package called **java.util.concurrent**

Using the classes of this package, thread-management is handed over to JVM instead of the developers taking care of it.

The top-level interfaces of concurrency are:

1. Executor
2. ExecutorService
3. ScheduledExecutorService

ScheduledExecutorService -> ExecutorService -> Executor

**package com.wipro.app;**

**import java.util.concurrent.Executor;**

**import java.util.concurrent.Executors;**

**public class ExecutorDemo {**

**public static void main(String[] args) {**

**Executor executor = Executors.*newSingleThreadExecutor*();**

**/\***

**\* pass Runnable object**

**\***

**\* Thread[pool-1-thread-1,5,main]**

**\*/**

**executor.execute(new Runnable() {**

**@Override**

**public void run() {**

**System.*out*.println(Thread.*currentThread*());**

**}**

**});**

**}**

**}**

**—----------------------------------------------------------------------------------------------------------**

**package com.wipro.service;**

**public class Summing implements Runnable{**

**private Long counter=0L;**

**public Summing() {**

**}**

**public Summing(Long counter) {**

**this.counter=counter;**

**}**

**@Override**

**public void run() {**

**Long sum=0L;**

**for(long i=1;i<=counter;i++) {**

**sum = sum + i;**

**}**

**System.*out*.println(sum);**

**}**

**@Override**

**public String toString() {**

**return "Summing [counter=" + counter + "]";**

**}**

**}**

**package com.wipro.app;**

**import java.util.concurrent.ExecutorService;**

**import java.util.concurrent.Executors;**

**import java.util.concurrent.TimeUnit;**

**import com.wipro.service.Summing;**

**public class SummingDemo {**

**public static void main(String[] args) {**

**ExecutorService service = Executors.*newFixedThreadPool*(10);**

**for(int i = 1;i<= 500;i++) {**

**Runnable runnable = new Summing(100000L+i);**

**service.execute(runnable);**

**}**

**service.shutdown();**

**try {**

**service.awaitTermination(5, TimeUnit.*MINUTES*);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**—-------------------------------------------**

**package com.wipro.app;**

**import java.util.Iterator;**

**import java.util.List;**

**import java.util.concurrent.ExecutorService;**

**import java.util.concurrent.Executors;**

**import java.util.concurrent.TimeUnit;**

**import com.wipro.service.Summing;**

**public class SummingDemo {**

**public static void main(String[] args) {**

**ExecutorService service = Executors.*newFixedThreadPool*(10);**

**for(int i = 1;i<= 500;i++) {**

**Runnable runnable = new Summing(100000L+i);**

**service.execute(runnable);**

**}**

**// service.shutdown();**

**List<Runnable> list = service.shutdownNow();**

**Iterator<Runnable> iterator = list.iterator();**

**while(iterator.hasNext()) {**

**System.*out*.println(iterator.next());**

**}**

**try {**

**service.awaitTermination(5, TimeUnit.*MINUTES*);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**—---------------------------------------------------------------------------------**

**Callable<T> interface**

**public interface Callable<T>{**

**T call() throws Exception;**

**}**

**package com.wipro.service;**

**import java.util.Random;**

**import java.util.concurrent.Callable;**

**public class Task implements Callable<Integer>{**

**@Override**

**public Integer call() throws Exception {**

**Thread.*sleep*(100);**

**return new Random().nextInt();**

**}**

**}**

**package com.wipro.app;**

**import java.util.Random;**

**import java.util.concurrent.Callable;**

**import java.util.concurrent.ExecutionException;**

**import java.util.concurrent.ExecutorService;**

**import java.util.concurrent.Executors;**

**import java.util.concurrent.Future;**

**import com.wipro.service.Task;**

**/\***

**\* The instance of the class that implements Callable interface is called**

**\* Callable object**

**\*/**

**public class FutureDemo {**

**public static void main(String[] args) throws InterruptedException {**

**ExecutorService service = Executors.*newFixedThreadPool*(10);**

**Future<Integer> future = service.submit(new Task());**

**while(!future.isDone()) {**

**// Thread.sleep(10);**

**System.*out*.println("Processing...");**

**}**

**Integer result;**

**try {**

**result = future.get();**

**System.*out*.println(result);**

**} catch (InterruptedException e) {**

**e.printStackTrace();**

**} catch (ExecutionException e) {**

**e.printStackTrace();**

**}**

**//Anonymous inner class**

**Future<Integer> futureNew = service.submit(new Callable<Integer>() {**

**@Override**

**public Integer call() throws Exception {**

**return new Random().nextInt();**

**}**

**});**

**while(!futureNew.isDone()) {**

**System.*out*.println("Working on it...");**

**}**

**try {**

**Integer resultNew = futureNew.get();**

**System.*out*.println(resultNew);**

**} catch (InterruptedException | ExecutionException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**—--------------------------------------------------------------------------------**

**Implement a task called, factorial of a given number using**

1. **Runnable interface**
2. **Callable interface**

**Using Concurrency classes and interfaces**

**Note: Do not create separate implementation classes instead use anonymous inner classes.**

**package com.wipro.app;**

**import java.util.concurrent.Callable;**

**import java.util.concurrent.ExecutionException;**

**import java.util.concurrent.ExecutorService;**

**import java.util.concurrent.Executors;**

**import java.util.concurrent.Future;**

**public class FactorialDemo {**

**public static void main(String[] args) throws InterruptedException, ExecutionException {**

**ExecutorService service = Executors.*newSingleThreadExecutor*();**

**//service.execute()**

**service.submit(new Runnable() {**

**@Override**

**public void run() {**

**long fact=1;**

**int n=5;**

**while(n>0) {**

**fact \*= n;**

**n--;**

**}**

**System.*out*.println(fact);**

**}**

**});**

**Future<Long> future = service.submit(new Callable<Long>() {**

**@Override**

**public Long call() throws Exception {**

**long fact=1;**

**int n=5;**

**while(n>0) {**

**fact \*= n;**

**n--;**

**}**

**return fact;**

**}**

**});**

**while(!future.isDone()) {**

**Thread.*sleep*(1);**

**}**

**System.*out*.println(future.get());**

**service.shutdown();**

**}**

**}**

**—------------------------------------------------------------------**

**A modifiable object is called mutable object and unmodifiable object is called an immutable object.**

**Ex. String objects are immutable objects whereas StringBuffer and StringBuilder objects mutable.**

**Immutable objects are always thread-safe. Since they read-only objects, no thread can make any change to it, hence immutable objects are thread-safe.**

**Even though StringBuffer objects are mutable they are thread-safe because the StringBuffer methods such as insert(),delete(),append() are**

**Synchronized.**

**When building user-defined classes, if possible build them as immutable classes or define the methods as synchronized.**

**This way user-defined objects are always thread-safe.**

**—----------------------------------------------------------------------------------------**

**Java Generics**

Generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods.

Ex. Comparable<Person>, Comparable<Integer>

Types can be built-in types ex. Integer, String etc or can be user-defined ex. Person, Account, Supplier etc.

Type parameters provide a way for you to re-use the same code with different inputs.

**Ex. Comparable<T>**

**public interface COMPARABLE<T>{**

**int compareTo(T o) ;**

**}**

**Comparable<Person>, Comparable<Integer>, Comparable<Box> etc.**

**Advantages:**

Code that uses generics has many benefits over non-generic code:

* **Stronger type checks at compile time.**  
   A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.

**Elimination of casts.** The following code snippet without generics requires casting:  
 *//Non-generic*

List list = new ArrayList();

list.add("hello");

list.add(10);

String s = **(String)** list.get(0);

Int x = **(Integer)**list.get(1);

When re-written to use generics, the code does not require casting:  
 //Generic

List<String> list = new ArrayList<String>();

list.add("hello");

list.add(10); //compiler error

String s = list.get(0); // no cast

# **Generic Types**

A *generic type* is a generic class or interface that is parameterized over types.

Ex.

Comparable<T>, Box<T>, Pair<K,V>

**//Non-generic**

public class BoxInt{

private Integer length;

private Integer width;

private Integer height;

}

BoxInt myBox= new BoxInt();

In the definition, The dimensions of the Box are restricted to only Integer type.

If I want the Box to have dimensions in real type?

public class BoxDouble{

private Double length;

private Double width;

private Double height;

}

BoxDouble myBox = new BoxDouble();

Now I want dimensions to be in Long,

**Solution: Go for Generics**

**//Generic class**

public class Box<T>{

private T length;

private T width;

private T height;

}

Box<Integer> myBox1 = new Box<>();

Box<Double> myBox2 = new Box<>();

Box<Long> myBox3 = new Box<>();

—-----------------------------------------------------------

**package** com.wipro.service;

**public** **class** Box<T> {

**private** T length;

**private** T width;

**private** T height;

**public** Box() {

}

**public** Box(T length, T width, T height) {

**super**();

**this**.length = length;

**this**.width = width;

**this**.height = height;

}

**public** T getLength() {

**return** length;

}

**public** **void** setLength(T length) {

**this**.length = length;

}

**public** T getWidth() {

**return** width;

}

**public** **void** setWidth(T width) {

**this**.width = width;

}

**public** T getHeight() {

**return** height;

}

**public** **void** setHeight(T height) {

**this**.height = height;

}

@Override

**public** String toString() {

**return** "Box [length=" + length + ", width=" + width + ", height=" + height + "]";

}

}

**package** com.wipro.app;

**import** com.wipro.service.Box;

**public** **class** BoxDemo {

**public** **static** **void** main(String[] args) {

Box<Integer> box1= **new** Box<>(5,5,5);

Box<Double> box2= **new** Box<>(7.2,5.6,9.0);

System.***out***.println("Volume of box1 = "+ box1.getLength()\*box1.getWidth()\*box1.getHeight() );

System.***out***.println("Volume of box2 = "+ box2.getLength()\*box2.getWidth()\*box2.getHeight());

}

}

—-----------------------------------------------------

//generic interface

**package** com.wipro.service;

/\*

\* Generic Interface

\*/

**public** **interface** Pair<K,V> {

K getKey();

V getValue();

}

**package** com.wipro.service;

/\*\*

\* Generic class

\*/

**public** **class** OrderedPair<K,V> **implements** Pair<K,V> {

**private** K key;

**private** V value;

**public** OrderedPair() {

}

**public** OrderedPair(K key, V value) {

**super**();

**this**.key = key;

**this**.value = value;

}

@Override

**public** K getKey() {

**return** **this**.key;

}

@Override

**public** V getValue() {

**return** **this**.value;

}

@Override

**public** String toString() {

**return** "OrderedPair [key=" + key + ", value=" + value + "]";

}

}

**package** com.wipro.app;

**import** com.wipro.service.OrderedPair;

**public** **class** OrderedPairDemo {

**public** **static** **void** main(String[] args) {

OrderedPair<Integer,Integer> orderedPair1 = **new** OrderedPair<>(4,4);

OrderedPair<Integer, Long> orderedPair2 = **new** OrderedPair<>(4,4L);

OrderedPair<Double, Double> orderedPair3 = **new** OrderedPair<>(4.0,4.0);

System.***out***.println(orderedPair1);

System.***out***.println(orderedPair2);

System.***out***.println(orderedPair3);

}

}

—-----------------------------------------------------------------------------------

## **Type Parameter Naming Conventions**

By convention, type parameter names are single, uppercase letters. Without this convention, it would be difficult to tell the difference between a type variable and an ordinary class or interface name.

The most commonly used type parameter names are:

* E - Element (used extensively by the Java Collections Framework)
* K - Key
* N - Number
* T - Type
* V - Value
* S,U,V etc. - 2nd, 3rd, 4th types

# **Generic Methods**

*Generic methods* are methods that introduce their own type parameters. This is similar to declaring a generic type, but the type parameter's scope is limited to the method where it is declared.

Static and non-static generic methods are allowed, as well as generic class constructors.

The syntax for a generic method includes a list of type parameters, inside angle brackets, which appears before the method's return type.

For static generic methods, the type parameter section must appear before the method's return type.

**package** com.wipro.service;

/\*\*

\* Generic Method

\* syntax:

\* public static <T> returnType methodName(T t1, T t2)

\*

\* Ex.

\* public static <K,V> boolean compare(OrderedPair<K,V> p1, OrderedPair<K,V> p2)

\*/

**public** **class** Utility {

**public** **static** <K,V> **boolean** compare(OrderedPair<K,V> p1, OrderedPair<K,V> p2) {

**return** p1.getKey().equals(p2.getKey()) && p1.getValue().equals(p2.getValue());

}

}

**package** com.wipro.app;

**import** com.wipro.service.OrderedPair;

**import** com.wipro.service.Utility;

**public** **class** UtilityTest {

**public** **static** **void** main(String[] args) {

OrderedPair<Integer,String> p1 = **new** OrderedPair<>(1,"Apple");

OrderedPair<Integer,String> p2 = **new** OrderedPair<>(2,"Banana");

OrderedPair<Integer,String> p3 = **new** OrderedPair<>(1,"Apple");

System.***out***.println(Utility.*compare*(p1, p2));

System.***out***.println(Utility.*compare*(p2, p3));

System.***out***.println(Utility.*compare*(p1, p3));

}

}

—-----------------------------------------------------------

# **Bounded Type Parameters**

There may be times when you want to restrict the types that can be used as type arguments in a parameterized type.

**Bounded type parameters restrict the types that can be used as type arguments.**

For example, a method that operates on numbers might only want to accept instances of Number or its subclasses. This is what *bounded type parameters* are for.

There are two types of bounds in Java Generics:

* **Upper Bound**: Specifies that the type parameter must be a subtype of a specific type.
* **Lower Bound**: Specifies that the type parameter must be a supertype of a specific type.

## **Upper Bounded Type Parameters**

Upper bounded type parameters restrict the type parameter to be a specific type or its subclasses. The extends keyword is used to define an upper bound.

**<T extends TypeName>**

Ex.

public static **<T extends Number>** void printDoubleValue(T value) {

System.out.println(value.doubleValue());

}

**<T extends Number>**

Object

String **Number**  Person

**Byte Short Integer Long Float Double**

Note: Here T can be Number or its subtypes i.e Byte, Short,Integer,Long,Float,Double but T cannot be String, Person , Object

**package** com.wipro.app;

**public** **class** UpperBoundedExample {

// Generic method with upper bounded type parameter

**public** **static** <T **extends** Number> **void** printDoubleValue(T value) {

System.***out***.println(value.doubleValue());

}

// Generic method with upper bounded type parameter

//T can be only String since there is no sub type for String

**public** **static** <T **extends** String> **void** printStringValue(T value) {

System.***out***.println(value);

}

**public** **static** **void** main(String[] args) {

*printDoubleValue*(10); // Integer

*printDoubleValue*(3.14); // Double

*printDoubleValue*(5.67f); // Float

*printDoubleValue*(5L); // Long

/\*

\* The method printDoubleValue(T) in the

\* type UpperBoundedExample is not applicable for the arguments (String)

\*/

// printDoubleValue("Hello");

*printStringValue*("Hello");

/\*

\* StringBuffer is not a sub class of String

\*/

// printStringValue(new StringBuffer("Hello"));

}

}

## **Lower Bounded Type Parameters**

Lower bounded type parameters restrict the type parameter to be a specific type or its supertypes. The super keyword is used to define a lower bound.

<T super TypeName>

Ex.

<T super Integer>

T can be Integer, Number or Object

Can T be Long or Double ? NO

**Wild Card(?)**

Wildcards in Method Parameters

Wildcards can also be used as method parameters. This allows us to write methods that can work with collections of different types.

Example:

public void printList(List<?> list) {

for (Object elem : list)

System.out.println(elem + " ");

System.out.println();

}

Ex.

**package** com.wipro.app;

**import** java.util.ArrayList;

**import** java.util.List;

**public** **class** GenericListDemo {

**public** **static** **void** main(String[] args) {

List<Integer> myList1= **new** ArrayList<>();

myList1.add(10); myList1.add(20); myList1.add(30);

*printList*(myList1);

List<String> myList2= **new** ArrayList<>();

myList2.add("Java");myList2.add("Java EE");

myList2.add("Javascript");

*printList*(myList2);

}

//Wild Card ? is unbounded , no restriction

**public** **static** **void** printList(List<?> list) {

**for** (Object elem : list)

System.***out***.println(elem + " ");

System.***out***.println();

}

}

**package com.wipro.app;**

**import java.util.ArrayList;**

**import java.util.List;**

**public class ExtendsSuperDemo {**

**public static void main(String[] args) {**

**List<Integer> list1= new ArrayList<>();**

**List<Number> list2= new ArrayList<>();**

**List<Long> list3= new ArrayList<>();**

**List<String> list4= new ArrayList<>();**

**List<Object> list5= new ArrayList<>();**

***printValues*(list1);**

***printValues*(list2);**

***printValues*(list3);**

***printNonNumericValues*(list4);**

***printNonNumericValues*(list5);**

**}**

**//super is lower-bound, String and above**

**private static void printNonNumericValues(List<? super String> list) {**

**for(int i=0;i<list.size();i++) {**

**System.*out*.println(list.get(i));**

**}**

**}**

**//extends is upper-bound, Number and below**

**private static void printValues(List<? extends Number> list) {**

**for(int i=0;i<list.size();i++) {**

**System.*out*.println(list.get(i));**

**}**

**}**

**}**

**—-------------------------------------------**

**Date API**

**Legacy Classes:** < jdk 1.8

Date, Calendar, GregorianCalendar, SimpleDateFormat

**Date API introduced in JDK 8**

LocalDate, LocalTime, LocalDateTime,

DateTimeFormatter, Period

| **Letter** | **Date or Time Component** | **Presentation** | **Examples** |
| --- | --- | --- | --- |
| G | Era designator | [Text](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#text) | AD |
| y | Year | [Year](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#year) | 1996; 96 |
| Y | Week year | [Year](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#year) | 2009; 09 |
| M | Month in year (context sensitive) | [Month](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#month) | July; Jul; 07 |
| L | Month in year (standalone form) | [Month](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#month) | July; Jul; 07 |
| w | Week in year | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 27 |
| W | Week in month | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 2 |
| D | Day in year | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 189 |
| d | Day in month | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 10 |
| F | Day of week in month | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 2 |
| E | Day name in week | [Text](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#text) | Tuesday; Tue |
| u | Day number of week (1 = Monday, ..., 7 = Sunday) | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 1 |
| a | Am/pm marker | [Text](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#text) | PM |
| H | Hour in day (0-23) | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 0 |
| k | Hour in day (1-24) | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 24 |
| K | Hour in am/pm (0-11) | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 0 |
| h | Hour in am/pm (1-12) | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 12 |
| m | Minute in hour | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 30 |
| s | Second in minute | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 55 |
| S | Millisecond | [Number](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#number) | 978 |
| z | Time zone | [General time zone](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#timezone) | Pacific Standard Time; PST; GMT-08:00 |
| Z | Time zone | [RFC 822 time zone](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#rfc822timezone) | -0800 |
| X | Time zone | [ISO 8601 time zone](https://docs.oracle.com/javase/8/docs/api/java/text/SimpleDateFormat.html#iso8601timezone) | -08; -0800; -08:00 |

—--------------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.text.DateFormat;

**import** java.text.ParseException;

**import** java.text.SimpleDateFormat;

**import** java.util.Calendar;

**import** java.util.Date;

**import** java.util.GregorianCalendar;

**import** java.util.Scanner;

/\*

\* Leagacy Date API classes are placed in java.util package

\* Java EPOCH:

\* This is the base time line in Java i.e all date and time related

\* calculations are based on this timeline.

\* 01-01-1970 00:00:00 GMT

\*

\* So 1st January 1970 mid-night 12 AM is Java epoch

\*

\* Calendar is an abstract class and GregorianCalendar is the concrete

\* sub class of Calendar

\*/

**public** **class** DateDemo {

**private** **static** Scanner *scanner* = **new** Scanner(System.***in***);

**public** **static** **void** main(String[] args) {

Date today = **new** Date();

//Thu Jul 11 14:50:11 IST 2024

System.***out***.println(today);

//pass time in milliseconds(1000 ms = 1s)

Date someDate = **new** Date(6789876789989L);

//01-01-1970 00:00:00 + 6789876789989L

//Mon Feb 28 18:23:09 IST 2185

System.***out***.println(someDate);

**try** {

//convert String -> Date

System.***out***.println("Enter birthdate(dd-mm-yyyy): ");

String dateAsString = *scanner*.nextLine();

DateFormat dateFormat = **new** SimpleDateFormat("dd-MM-yyyyy");

Date birthdate = dateFormat.parse(dateAsString);

System.***out***.println(birthdate);

} **catch** (ParseException e) {

e.printStackTrace();

}

Calendar hiredate = **new** GregorianCalendar(2022,10,15);

System.***out***.println(hiredate.get(Calendar.***DATE***));

}

}

—----------------------------------------------------------------

**package** com.wipro.app;

**import** java.time.LocalDate;

**import** java.time.LocalDateTime;

**import** java.time.LocalTime;

**import** java.time.Period;

**import** java.time.format.DateTimeFormatter;

**import** java.util.Scanner;

/\*\*

\* LocalDate, LocalTime, LocalDateTime

\* DateTimeFormatter, Period

\*

\* The above classes are part of java.time package

\*/

**public** **class** NewDateAPI {

**private** **static** Scanner *scanner* = **new** Scanner(System.***in***);

**public** **static** **void** main(String[] args) {

LocalDate today = LocalDate.*now*();

//2024-07-11

System.***out***.println(today);

LocalTime currentTime = LocalTime.*now*();

//15:15:30.500429340

System.***out***.println(currentTime);

LocalDateTime dateAndTime = LocalDateTime.*now*();

//2024-07-11T15:16:34.447399053

System.***out***.println(dateAndTime);

LocalDate hiredate = LocalDate.*of*(2022, 10, 25);

System.***out***.println(hiredate);

LocalDateTime birthdate =

LocalDateTime.*of*(2022,1,1, 10,15,30);

System.***out***.println(birthdate);

//String -> LocalDate

System.***out***.println("Enter marriage date:(yyyy-mm-dd): ");

String marriageDateAsString = *scanner*.nextLine();

LocalDate marriageDate = LocalDate.*parse*(marriageDateAsString);

System.***out***.println(marriageDate);

System.***out***.println("Enter exam date: (dd/mm/yyyy)");

String examAsString = *scanner*.nextLine();

DateTimeFormatter formatter =

DateTimeFormatter.*ofPattern*("dd/MM/yyyy");

LocalDate examDate = LocalDate.*parse*(examAsString,formatter);

System.***out***.println(examDate);

Period period = marriageDate.until(today);

System.***out***.println(period.getYears()+" years,"+ period.getMonths()+" months, "+period.getDays()+" days");

**if**(marriageDate.isAfter(today)) {

System.***out***.println("Marriage date is in near future");

}**else** {

System.***out***.println("Already married");

}

}

}

—--------------------------------------------------------

**Shipment class** : com.wipro.model package

shipmentId, shipmentDate, shipmentTo, orderId,orderDate, deliveredDate, status

**Validator class** : com.wipro.service package

public Boolean isValidShipment(Shipment shipment):

false if shipmentDate is before orderDate or shipmentDate is after deliveredDate else return true

Valid dates:

Ex. shipmentDate: 2024-07-11

orderDate: 2024-07-10

deliveredDate: 2024-07-11 or after

**Tester class**: com.wipro.app package

Test the above classes

—----------------------------------------------------------------

**package** com.wipro.model;

**public** **enum** StatusTyp {

***DELIVERED***, ***INTRANSIT***, ***DAMAGED***,***LOST***

}

**package** com.wipro.model;

**import** java.time.LocalDate;

/\*\*

\* Shipment class : com.wipro.model package

shipmentId, shipmentDate, shipmentTo, orderId,orderDate,

deliveredDate, status

Validator class : com.wipro.service package

public Boolean isValidShipment(Shipment shipment):

false if shipmentDate is before orderDate or shipmentDate is after deliveredDate else return true

Valid dates:

Ex. shipmentDate: 2024-07-11

orderDate: 2024-07-10

deliveredDate: 2024-07-11 or after

Tester class: com.wipro.app package

Test the above classes

\*/

**public** **class** Shipment {

**private** Long shipmentId;

**private** LocalDate shipmentDate;

**private** String shipmentTo;

**private** Long orderId;

**private** LocalDate orderDate;

**private** LocalDate deliveredDate;

**private** StatusTyp status;

**public** Shipment() {

}

**public** Shipment(Long shipmentId, LocalDate shipmentDate, String shipmentTo, Long orderId, LocalDate orderDate,

LocalDate deliveredDate, StatusTyp status) {

**super**();

**this**.shipmentId = shipmentId;

**this**.shipmentDate = shipmentDate;

**this**.shipmentTo = shipmentTo;

**this**.orderId = orderId;

**this**.orderDate = orderDate;

**this**.deliveredDate = deliveredDate;

**this**.status = status;

}

**public** Long getShipmentId() {

**return** shipmentId;

}

**public** **void** setShipmentId(Long shipmentId) {

**this**.shipmentId = shipmentId;

}

**public** LocalDate getShipmentDate() {

**return** shipmentDate;

}

**public** **void** setShipmentDate(LocalDate shipmentDate) {

**this**.shipmentDate = shipmentDate;

}

**public** String getShipmentTo() {

**return** shipmentTo;

}

**public** **void** setShipmentTo(String shipmentTo) {

**this**.shipmentTo = shipmentTo;

}

**public** Long getOrderId() {

**return** orderId;

}

**public** **void** setOrderId(Long orderId) {

**this**.orderId = orderId;

}

**public** LocalDate getOrderDate() {

**return** orderDate;

}

**public** **void** setOrderDate(LocalDate orderDate) {

**this**.orderDate = orderDate;

}

**public** LocalDate getDeliveredDate() {

**return** deliveredDate;

}

**public** **void** setDeliveredDate(LocalDate deliveredDate) {

**this**.deliveredDate = deliveredDate;

}

**public** StatusTyp getStatus() {

**return** status;

}

**public** **void** setStatus(StatusTyp status) {

**this**.status = status;

}

@Override

**public** String toString() {

**return** "Shipment [shipmentId=" + shipmentId + ", shipmentDate=" + shipmentDate + ", shipmentTo=" + shipmentTo

+ ", orderId=" + orderId + ", orderDate=" + orderDate + ", deliveredDate=" + deliveredDate + ", status="

+ status + "]";

}

}

**package** com.wipro.service;

**import** com.wipro.model.Shipment;

/\*

\* false if shipmentDate is before orderDate or

\* shipmentDate is after deliveredDate else return true

\*/

**public** **class** Validator {

**public** Boolean isValidShipment(Shipment shipment) {

**if**(shipment.getShipmentDate().isBefore(shipment.getOrderDate()) ||

shipment.getShipmentDate().isAfter(shipment.getDeliveredDate())

) {

**return** **false**;

}

**return** **true**;

}

}

**package** com.wipro.app;

**import** java.time.LocalDate;

**import** com.wipro.model.Shipment;

**import** com.wipro.model.StatusTyp;

**import** com.wipro.service.Validator;

**public** **class** ShipmentTester {

**public** **static** **void** main(String[] args) {

Shipment shipment1 = **new** Shipment(123456L,

LocalDate.*of*(2024, 07, 11),

"Smith",

56768L,LocalDate.*of*(2024, 07, 10),

LocalDate.*of*(2024, 07, 11),

StatusTyp.***DELIVERED***

);

Validator validator = **new** Validator();

**if**(validator.isValidShipment(shipment1)) {

System.***out***.println(shipment1);

}**else** {

System.***out***.println("Invalid Shipment Details");

}

}

}

—------------------------------------------------------

**Lambda Expression**

Lambda expression represents an **instance of functional interface**

A lambda expression is an anonymous block of code that

encapsulates an expression or a block of statements and returns a result

*Syntax of Lambda expression:*

**(argument list) -> { implementation }**

The arrow operator -> is used to separate list of parameters and

body of lambda expression.

**package** com.wipro.service;

@FunctionalInterface

**public** **interface** MaxFinder {

**public** **abstract** **int** maxmimum(**int** num1, **int** num2);

}

**package** com.wipro.app;

**import** com.wipro.service.MaxFinder;

/\*\*

\* Lambda expression represents an instance of functional interface

\*

\* InterfaceType objectName = (arguments) -> {implementation }

\*

\*

\*/

**public** **class** LambdaDemo {

**public** **static** **void** main(String[] args) {

MaxFinder maxFinder = (**int** n1, **int** n2) -> {

**if**(n1>n2) {

**return** n1;

}**else** {

**return** n2;

}

};

System.***out***.println(maxFinder.maxmimum(1029,1029));

MaxFinder maxFinderNew = (n1,n2) -> (n1>n2)?n1:n2;

System.***out***.println(maxFinderNew.maxmimum(10,15));

}

}

—-------------------------------------

Create functional interface, **ISumOfSquare** that returns sum of squares of all the digits from 1 to given number and test the method using Lambda expression.

**public abstract long sumOfSquares(int num)**

Create functional interfaces for the following methods and implement using Lambda expressions

**IFactorial**

1. public abstract long factorial(int num); returns factorial of given number

**IFactorial** iFact = n -> {

long fact=1;

` while(n>0){

fact \*= n–;

}

return fact;

};

System.out.println(iFact.factorial(5));

**IGreeting**

1. public abstract void greeting(String name): : prints “Good Day, <name>”

**IGreeting** iGreet = str -> “Good Day,”+str;

System.out.println(iGreet.greeting(“Smith”);

**IEven**

3. public abstract boolean isEven(int num); returns if given number is even

number else returns false

**IEven** iEven = n -> (n%2==0)?true:false;

System.out.println(iEven.isEven(5));

**IDate**

4. public abstract String getDateTime() ; returns current date and time as a

String

**IDate** iDate = ()-> LocalDateTime.*now*().toString()

System.out.println(iDate.getDateTime());

—--------------------------------------------------------------------------------------------

**package** com.wipro.service;

@FunctionalInterface

**public** **interface** ISumOfSquare {

**public** **abstract** **long** sumOfSquares(**int** num);

}

**package** com.wipro.app;

**import** com.wipro.service.ISumOfSquare;

**import** com.wipro.service.MaxFinder;

/\*\*

\* Lambda expression represents an instance of functional interface

\*

\* InterfaceType objectName = (arguments) -> {implementation }

\*

\*

\*/

**public** **class** LambdaDemo {

**public** **static** **void** main(String[] args) {

MaxFinder maxFinder = (**int** n1, **int** n2) -> {

**if**(n1>n2) {

**return** n1;

}**else** {

**return** n2;

}

};

System.***out***.println(maxFinder.maxmimum(1029,1029));

MaxFinder maxFinderNew = (n1,n2) -> (n1>n2)?n1:n2;

System.***out***.println(maxFinderNew.maxmimum(10,15));

ISumOfSquare iSum = (n) ->{

**long** sum=0;

**for**(**int** i=1;i<=n;i++) {

sum = sum + i\*i;

}

**return** sum;

};

System.***out***.println(iSum.sumOfSquares(10));

}

}

—-------------------------------------------------------------------------------

**Method Declaration combinations:**

1. **void methodName();**

Ex.

void greeting(){

System.out.println(“Hello World!”);

}

1. **returnType methodName();**

Ex.

String greeting(){

return “Hello World!”;

}

1. **void methodName(<parameter(s)>)**

Ex.

void greeting(String name){

System.out.println(“Hello, “+ name);

}

1. **returnType methodName(<parameter(s)>);**

Ex.

String greeting(String name){

return “Hello, “+ name;

}

**Built-in Functional interfaces**

Following are the main built-in Functional Interfaces

1. **Supplier**
2. **Consumer**
3. **Predicate**
4. **Function**

**Supplier<T>**

**T get()**

**Consumer<T>**

**void accept(T t)**

**BiConsumer<T,U>**

**void accept(T t, U u)**

**Predicate<T>**

**boolean test(T t)**

**BiPredicate<T,U>**

**boolean test(T t, U u)**

**Function<T,R>**

**R apply(T t)**

**BiFunction<T,U,R>**

**R apply(T t, U u)**

**—-------------------------------------------------------------------------------------------**

**Function<Integer,Long>**

*public abstract long factorial(int num); returns factorial of given number*

public abstract Long **apply**(Integer num); returns factorial of given number

**Consumer<String>**

*public abstract void greeting(String name): : prints “Good Day, <name>”*

public abstract void **accep**t(String name): : prints “Good Day, <name>”

**Predicate<Integer>**

*public abstract boolean isEven(int num); returns if given number is even*

*number else returns false*

public abstract Boolean **test**(Integer num);

**Supplier<String>**

public abstract String getDateTime() ; returns current date and time as a

String

public abstract String **get**()

—----------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.util.function.Function;

/\*\*

\* Function<Integer,Long>

public abstract long factorial(int num); returns factorial of given number

public abstract Long apply(Integer num); returns factorial of given number

Consumer<String>

public abstract void greeting(String name): : prints “Good Day, <name>”

public abstract void accept(String name): : prints “Good Day, <name>”

Predicate<Integer>

public abstract boolean isEven(int num); returns if given number is even

number else returns false

public abstract Boolean test(Integer num);

Supplier<String>

public abstract String getDateTime() ; returns current date and time as a

String

public abstract String get()

\*/

**public** **class** BFITester {

**public** **static** **void** main(String[] args) {

/\* Implement factorial method that receives an integer value

\* and returns long value using built-in functional interface

\*

\*/

Function<Integer,Long> fun1 = n->{

**long** f=1;

**while**(n>0) {

f=f\*n--;

}

**return** f;

};

System.***out***.println("Factorial of 5 is "+ fun1.apply(5));

System.***out***.println("------------------------");

Consumer<String> con1 = s -> System.***out***.print("Good Day, "+s);

con1.accept("Smith");

System.***out***.println("\n------------------------");

Predicate<Integer> pred1 = n -> (n%2==0)?**true**:**false**;

System.***out***.println(pred1.test(5));

System.***out***.println("-------------------------");

Supplier<String> sup1 = () -> LocalDateTime.*now*().toString();

System.***out***.println(sup1.get());

}

}

—------------------------------------------------------------------------------------------------

Implement the following methods using Built-in functional interface

String method(String fname, String lname) : returns “fname name”

Double method(int lbound, int ubound) : returns of average of the

digits between lower bound and upper bound, inclusive of both

Boolean method(String userId,String password):

Returns true if userId and password are different else return false

void method(Person person): display person adharcard, name and birthdate in dd-mm-yyyy format and address.

—---------------------------------------------------------------------

**package** com.wipro.model;

**public** **enum** Gender {

***MALE***, ***FEMALE***, ***TRANSGENDER***

}

**package** com.wipro.model;

**import** java.time.LocalDate;

//POJO : Plain Old Java Object

**public** **class** Person **implements** Comparable<Person>{

**private** Long adharCard;

**private** String name;

**private** Gender gender;

**private** LocalDate birthdate;

**private** String address;

**private** Long mobile;

**public** Person() {

}

**public** Person(Long adharCard, String name, Gender gender, LocalDate birthdate, String address, Long mobile) {

**super**();

**this**.adharCard = adharCard;

**this**.name = name;

**this**.gender = gender;

**this**.birthdate = birthdate;

**this**.address = address;

**this**.mobile = mobile;

}

**public** Long getAdharCard() {

**return** adharCard;

}

**public** **void** setAdharCard(Long adharCard) {

**this**.adharCard = adharCard;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** Long getMobile() {

**return** mobile;

}

**public** **void** setMobile(Long mobile) {

**this**.mobile = mobile;

}

**public** Gender getGender() {

**return** gender;

}

**public** **void** setGender(Gender gender) {

**this**.gender = gender;

}

**public** LocalDate getBirthdate() {

**return** birthdate;

}

**public** **void** setBirthdate(LocalDate birthdate) {

**this**.birthdate = birthdate;

}

**public** String getAddress() {

**return** address;

}

**public** **void** setAddress(String address) {

**this**.address = address;

}

@Override

**public** String toString() {

**return** "Person [adharCard=" + adharCard + ", name=" + name + ", gender=" + gender + ", birthdate=" + birthdate

+ ", address=" + address + ", mobile=" + mobile + "]";

}

//p1.compareTo(p2)

@Override

**public** **int** compareTo(Person o) {

**return** **this**.getAdharCard().compareTo(o.getAdharCard());

}

}

—-------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.time.LocalDate;

**import** java.util.function.BiFunction;

**import** java.util.function.BiPredicate;

**import** java.util.function.Consumer;

**import** java.util.function.Function;

**import** com.wipro.model.Gender;

**import** com.wipro.model.Person;

/\*\*

\* String method(String fname, String lname) : returns “fname name”

Double method(int lbound, int ubound) : returns of average of the

digits between lower bound and upper bound, inclusive of both

Boolean method(String userId,String password):

Returns true if userId and password are different else return false

void method(Person person): display person adharcard, name and birthdate in

dd-mm-yyyy format and address.

\*/

**public** **class** FIDemo {

**public** **static** **void** main(String[] args) {

BiFunction<String,String,String> bi = (fname,lname)-> fname+" "+lname;

System.***out***.println(bi.apply("Rohit", "Sharma"));

BiFunction<Integer,Integer,Double> bi1 = (l,u)->{

**int** sum=0;

**for**(**int** i=l;i<=u;i++) {

sum += i;

}

**return** (**double**)sum/(u-l+1);

};

System.***out***.println(bi1.apply(10, 20));

BiPredicate<String,String> bip = (userid,pwd)-> !userid.equals(pwd);

System.***out***.println(bip.test("admin", "admin"));

System.***out***.println(bip.test("admin", "admin@123"));

Consumer<Person> cons = p -> System.***out***.println(p.getAdharCard()+","+ p.getName()

+ ","+ p.getBirthdate()+","+p.getAddress());

cons.accept(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L ));

// A method that receives Person object and returns Gender

Function<Person,Gender> fn = p -> p.getGender();

System.***out***.println(fn.apply(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L )));

}

}

Pre-defined Built-in Functional interfaces

Supplier:

IntSupplier, LongSupplier, DoubleSupplier etc.

Consumer:

IntConsumer, LongConsumer, DoubleConsumer etc.

Predicate:

IntPredicate, LongPredicate, DoubleConsumer etc.

Function:

IntFunction, LongFunction, DoubleFunction etc.

UnaryOpertor and BinaryOperator are two special type of built-in functional interface Function and BiFunction.

Ex.

Function<Integer,Integer>

Or

UnaryOperator<Integer>

Ex.

BiFunction<Integer,Integer,Integer>

Or

BinaryOperator<Integer>

@FunctionalInterface

public interface BinaryOperator<T> extends BiFunction<T,T,T>{

}

// a method that returns sum of 2 integers

—----------------------------------------

**package** com.wipro.app;

**import** java.time.LocalDate;

**import** java.util.function.BiFunction;

**import** java.util.function.BiPredicate;

**import** java.util.function.BinaryOperator;

**import** java.util.function.Consumer;

**import** java.util.function.Function;

**import** java.util.function.IntFunction;

**import** java.util.function.UnaryOperator;

**import** com.wipro.model.Gender;

**import** com.wipro.model.Person;

/\*\*

\* String method(String fname, String lname) : returns “fname name”

Double method(int lbound, int ubound) : returns of average of the

digits between lower bound and upper bound, inclusive of both

Boolean method(String userId,String password):

Returns true if userId and password are different else return false

void method(Person person): display person adharcard, name and birthdate in

dd-mm-yyyy format and address.

\*/

**public** **class** FIDemo {

**public** **static** **void** main(String[] args) {

BiFunction<String,String,String> bi = (fname,lname)-> fname+" "+lname;

System.***out***.println(bi.apply("Rohit", "Sharma"));

BiFunction<Integer,Integer,Double> bi1 = (l,u)->{

**int** sum=0;

**for**(**int** i=l;i<=u;i++) {

sum += i;

}

**return** (**double**)sum/(u-l+1);

};

System.***out***.println(bi1.apply(10, 20));

BiPredicate<String,String> bip = (userid,pwd)-> !userid.equals(pwd);

System.***out***.println(bip.test("admin", "admin"));

System.***out***.println(bip.test("admin", "admin@123"));

Consumer<Person> cons = p -> System.***out***.println(p.getAdharCard()+","+ p.getName()

+ ","+ p.getBirthdate()+","+p.getAddress());

cons.accept(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L ));

// A method that receives Person object and returns Gender

Function<Person,Gender> fn = p -> p.getGender();

System.***out***.println(fn.apply(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L )));

/\* Function<T,R>

\* Function<Integer,Long> lng= { };

\*

\* IntFunction<R>

\* IntFunctio<Long> lng = {};

\*/

//returns factorial of given number

IntFunction<Long> lng = n ->{

Long fact=1L;

**while**(n>0) {

fact \*= n--;

}

**return** fact;

};

System.***out***.println("Factorial of 5: "+ lng.apply(5));

// a method that returns sum of 2 integers

BiFunction<Integer,Integer,Integer> sm = (a,b)->a+b;

System.***out***.println(sm.apply(5, 6));

BinaryOperator<Integer> bop = (x,y)->x+y;

System.***out***.println(bop.apply(5, 6));

// n=n+5

UnaryOperator<Integer> uop = n->n+5;

System.***out***.println(uop.apply(5));

}

}

—------------------------------------------

**Method Reference operator( ::)**

This is another way of invoking a method.

It is a new way to refer a method by its name instead of calling it directly

Replacing Lambda with method reference

Ex.

**Consumer<String> consumer = (String str) -> System.out.println(str);**

**OR**

**Consumer<String> consumer = System.out::println ;**

—-------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.time.LocalDate;

**import** java.util.function.BiFunction;

**import** java.util.function.BiPredicate;

**import** java.util.function.BinaryOperator;

**import** java.util.function.Consumer;

**import** java.util.function.Function;

**import** java.util.function.IntFunction;

**import** java.util.function.UnaryOperator;

**import** com.wipro.model.Gender;

**import** com.wipro.model.Person;

/\*\*

\* String method(String fname, String lname) : returns “fname name”

Double method(int lbound, int ubound) : returns of average of the

digits between lower bound and upper bound, inclusive of both

Boolean method(String userId,String password):

Returns true if userId and password are different else return false

void method(Person person): display person adharcard, name and birthdate in

dd-mm-yyyy format and address.

\*/

**public** **class** FIDemo {

**public** **static** **void** main(String[] args) {

BiFunction<String,String,String> bi = (fname,lname)-> fname+" "+lname;

System.***out***.println(bi.apply("Rohit", "Sharma"));

BiFunction<Integer,Integer,Double> bi1 = (l,u)->{

**int** sum=0;

**for**(**int** i=l;i<=u;i++) {

sum += i;

}

**return** (**double**)sum/(u-l+1);

};

System.***out***.println(bi1.apply(10, 20));

BiPredicate<String,String> bip = (userid,pwd)-> !userid.equals(pwd);

System.***out***.println(bip.test("admin", "admin"));

System.***out***.println(bip.test("admin", "admin@123"));

Consumer<Person> cons = p -> System.***out***.println(p.getAdharCard()+","+ p.getName()

+ ","+ p.getBirthdate()+","+p.getAddress());

cons.accept(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L ));

// A method that receives Person object and returns Gender

Function<Person,Gender> fn = p -> p.getGender();

System.***out***.println(fn.apply(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L )));

/\* Function<T,R>

\* Function<Integer,Long> lng= { };

\*

\* IntFunction<R>

\* IntFunctio<Long> lng = {};

\*/

//returns factorial of given number

IntFunction<Long> lng = n ->{

Long fact=1L;

**while**(n>0) {

fact \*= n--;

}

**return** fact;

};

System.***out***.println("Factorial of 5: "+ lng.apply(5));

// a method that returns sum of 2 integers

BiFunction<Integer,Integer,Integer> sm = (a,b)->a+b;

System.***out***.println(sm.apply(5, 6));

BinaryOperator<Integer> bop = (x,y)->x+y;

System.***out***.println(bop.apply(5, 6));

// n=n+5

UnaryOperator<Integer> uop = n->n+5;

System.***out***.println(uop.apply(5));

//Method Reference

System.***out***.println("Displaying all the details of a Person..");

Consumer<Person> cons1 = p -> System.***out***.println(p);

cons1.accept(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L ));

/\*

\* replacing

\* p -> System.out.println(p);

\* with

\* System.out::println;

\*/

Consumer<Person> cons2 = System.***out***::println;

cons2.accept(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L ));

}

}

—-----------------------------------------------------------------------------------

Java Reflection

Retrospecting Java classes and objects at runtime is called reflection.

There are classes built in **java.lang.reflect** package which can used to introspect the Java classes and objects at runtime.

In the object class, we have a method called, getClass() which returns the Class object which contains details of the class. With this method, we can know details the classes at runtime.

**package** com.wipro.app;

**import** java.lang.reflect.Field;

**import** java.time.LocalDate;

**import** com.wipro.model.Gender;

**import** com.wipro.model.Person;

**public** **class** ReflectionDemo {

**public** **static** **void** main(String[] args) {

String greeting = **new** String("Welcome to java reflection");

Class myClassRef = greeting.getClass();

System.***out***.println(myClassRef.getName());

Person person = **new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2002, 10, 12),"Hyderabad",987898879L );

Class myClass = person.getClass();

System.***out***.println(myClass.getName());

/\*

\* myClass.getFields() returns only public fields

\*/

Field[] fields = myClass.getDeclaredFields();

**for**(Field f: fields) {

System.***out***.println(f.getName());

}

}

}

—------------------------------------------------------------------------------------------

**Collection Framework**

* Part of java.util package
* The Collection implementations dynamically grow as we add objects to it and dynamically shrink as we remove objects from them

Top-level interface:

1. Collection interface

Sub interfaces:

List, Set and Queue

1. Map
2. Implementation classes of List interface:
3. **ArrayList**
4. LinkedList
5. *Vector*
6. *Stack*

2. Implementation classes of Set interface:

1. **HashSet**
2. **LinkedHashSet**
3. **TreeSet**

3. Implementation classes of Map interface

1. **HashMap**
2. **LinkedHashMap**
3. **TreeMap**

—---------------------------------------------------------------------------------------

**Stream API**

* Stream API methods are implemented on Arrays as well as Collection implementations.

List<Integer> marksList = new ArrayList<>();

marksList.add(89); marksList.add(59); marksList.add(99);

marksList.add(65); marksList.add(89); marksList.add(90);

Methods of displaying?

*//index-based*

for(int i=0 ; i<marksList.size();i++){

System.out.println(marksList.get(i));

}

*//Iterator*

Iterator<Integer> iterator = marksList.iterator();

while(iterator.hasNext() ){

System.out.println(iterator.next());

}

Both the above methods use loops.

This type of coding is called **imperative programming** wherecoders have to build logic to implement.

**Declarative Programming**

* **Also called of functional programming**

In this approach, instead of building the logic the coders will be invoking the methods to perform the task.

In imperative programming the developers need to know **what the task is** and **how** to achieve it.

In declarative programming the developers need to know **what i**s required to process the task, i.e need to know methods that would perform the task.

Ex.

marksList.forEach((m)->System.out.println(m));

or

marksList.forEach(System.out::println);

**Stream API methods help us to move towards functional programming.**

**package** com.wipro.app;

**import** java.util.ArrayList;

**import** java.util.Iterator;

**import** java.util.List;

**public** **class** ForEachDemo {

**public** **static** **void** main(String[] args) {

List<Integer> marksList = **new** ArrayList<>();

marksList.add(89); marksList.add(59); marksList.add(99);

marksList.add(65); marksList.add(89); marksList.add(90);

System.***out***.println("Imperative Programming...");

System.***out***.println("displaying the list through index-based... ");

**for**(**int** i=0 ; i<marksList.size();i++){

System.***out***.println(marksList.get(i));

}

System.***out***.println("displaying the list using iterator... ");

Iterator<Integer> iterator = marksList.iterator();

**while**(iterator.hasNext() ){

System.***out***.println(iterator.next());

}

System.***out***.println("Declarative programming...using lambda");

marksList.forEach(m->System.***out***.println(m) ) ;

System.***out***.println("Declarative programming...using method reference");

marksList.forEach(System.***out***::println);

}

}

**Stream API operations:**

Stream API methods can be applied only on Stream objects.

How to acquire a Stream object:

1. Stream.of()
2. Arrays.stream(arrrayName)
3. All the Collection framework implementations such as ArrayList, HashSet, HashMap etc have an instance method, stream() that return Stream object.

Stream API operations are categorized into

1. Intermediate operations: can be pipelined, ie. output of one intermediate operation can be input to another intermediate operation and so on.

map(), filter(), concat(), limit(), skip(), sorted()

1. Terminal operations: final operations that close the stream

forEach(), reduce(), min(), max(), count(), collect()

Stream API classes are part of java.util.stream package

—------------------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.stream.Stream;

/\*

\* forEach() is an terminal operation

\*/

**public** **class** StreamAPITester {

**public** **static** **void** main(String[] args) {

Stream<Integer> stream1 = Stream.*of*(10,20,25,16,9,48,21,57,80);

stream1.forEach(System.***out***::println);

/\*

\* Runtime error

\* stream has already been operated upon or closed

\*/

// stream1.forEach(System.out::println);

System.***out***.println("---------------------------------");

Integer[] marks = **new** Integer[] {10,20,25,16,9,48,21,57,80};

Stream<Integer> marksStream = Arrays.*stream*(marks);

marksStream.forEach(System.***out***::println);

System.***out***.println("---------------------------------");

Arrays.*stream*(marks).forEach(System.***out***::println);

System.***out***.println("---------------------------------");

//Integer[] -> ArrayList

List<Integer> marksList = Arrays.*asList*(marks);

System.***out***.println("Displaying: marksList.forEach()...");

marksList.forEach(System.***out***::println);

// Stream<Integer> marksListStream = marksList.stream();

// marksListStream.forEach(System.out::println);

System.***out***.println("Displaying: marksList.stream().forEach()...");

marksList.stream().forEach(System.***out***::println);

}

}

—----------------------------------------------------

**Intermediate operation: map()**

This is a transformation method

**<R> Stream<R> map(java.util.function.Function<? super T, ? extends R> mapper)**

Function<T,R>

R apply(T t) : receives an object of type, T and returns an object of type, R

**package** com.wipro.app;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**public** **class** IntermediateDemo {

**public** **static** **void** main(String[] args) {

Integer[] marks = **new** Integer[] {10,20,25,16,9,48,21,57,80};

List<Integer> marksList = Arrays.*asList*(marks);

marksList.stream().map(m->m\*m).forEach(System.***out***::println);

System.***out***.println("--------------------------");

// List<Integer> squaredList = marksList.stream()

// .map(n->n\*n)

// .collect(Collectors.toList());

List<Integer> squaredList = marksList.stream()

.map(n->n\*n)

.toList();

squaredList.forEach(System.***out***::println);

List<String> courseList= **new** ArrayList<>();

courseList.add("Java");courseList.add("java ee");

courseList.add("JavaScript");courseList.add("Spring");

//convert & collect the above course names to upper-case letter into another list and display

List<String> courseNewList = courseList.stream()

.map(s->s.toUpperCase())

.toList();

courseNewList.forEach(System.***out***::println);

}

}

**filter()**

[**Stream**](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)**<**[**T**](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)**>** [**filter**](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html#filter-java.util.function.Predicate-)**(**[**Predicate**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html)**<? super** [**T**](https://docs.oracle.com/javase/8/docs/api/java/util/stream/Stream.html)**> predicate)**

**Returns a stream consisting of the elements of this stream that match the given predicate.**

**Predicate<T>**

**boolean test(T t)**

**//display the courses with java in it**

courseList.stream()

.map(s->s.toUpperCase())

.filter(s->s.contains("JAVA"))

.forEach(System.*out*::println);

**Special type of filters:**

**Distinct :filter only non-duplicate object**

**limit(n) : Limit the stream to first n elements**

**skip(n) : Skip first n elements**

**—-------------------------------------------------**

**package com.wipro.app;**

**import java.util.ArrayList;**

**import java.util.Arrays;**

**import java.util.List;**

**public class IntermediateDemo {**

**public static void main(String[] args) {**

**Integer[] marks = new Integer[] {10,20,25,16,9,48,21,57,80};**

**List<Integer> marksList = Arrays.*asList*(marks);**

**marksList.stream().map(m->m\*m).forEach(System.*out*::println);**

**System.*out*.println("--------------------------");**

**// List<Integer> squaredList = marksList.stream()**

**// .map(n->n\*n)**

**// .collect(Collectors.toList());**

**List<Integer> squaredList = marksList.stream()**

**.map(n->n\*n)**

**.toList();**

**squaredList.forEach(System.*out*::println);**

**System.*out*.println("-------------------------------------");**

**List<String> courseList= new ArrayList<>();**

**courseList.add("Java");courseList.add("java ee");**

**courseList.add("JavaScript");courseList.add("Spring");**

**courseList.add("Java");courseList.add("java ee");**

**//convert & collect the above course names to upper-case letter into another list and display**

**List<String> courseNewList = courseList.stream()**

**.map(s->s.toUpperCase())**

**.toList();**

**courseNewList.forEach(System.*out*::println);**

**System.*out*.println("----------------------------");**

**//display the courses with java in it**

**courseList.stream()**

**.map(s->s.toUpperCase())**

**.filter(s->s.contains("JAVA"))**

**.forEach(System.*out*::println);**

**System.*out*.println("Distinct courses...");**

**courseList.stream().distinct().forEach(System.*out*::println);**

**System.*out*.println("Display first 3 courses..");**

**courseList.stream().limit(3).forEach(System.*out*::println);**

**System.*out*.println("Skip first 3 courses..");**

**courseList.stream().skip(3).forEach(System.*out*::println);**

**// display the courses in an order**

**courseList.stream().sorted().forEach(System.*out*::println);**

**/\***

**\* Optional<Integer> java.util.stream.Stream.min(**

**Comparator<? super Integer> comparator**

**)**

**Comparator<Integer> : int compare(Integer, Integer)**

**The compare() has to be implemented using lambda,**

**(a,b)->a.compareTo(b)**

**\*/**

**Optional<Integer> optional = marksList.stream().min((a,b)->a.compareTo(b));**

**//get() extract the result from the Optional object**

**if(optional.isPresent()) {**

**System.*out*.println("Minimum mark: "+optional.get());**

**}else {**

**System.*out*.println("No min value");**

**}**

**int maxMark = marksList.stream().max((a,b)->a.compareTo(b)).get();**

**System.*out*.println("Maximum mark: "+ maxMark);**

**}**

**}**

**—------------------------------------**

**reduce() method**

The reduce operation on streams, which repeatedly applies an operation on each element until a result is produced

**Optional<T> reduce(java.util.function.BinaryOperator<T> accumulator))**

**BinaryOperator is same as BiFunction<T,T,T>**

**T apply(T t1, T t2)**

**In reduce() method implement T apply(T t1, T t2) using lambda**

/\*

Optional<Integer> java.util.stream.Stream.reduce(

BinaryOperator<Integer> accumulator

)

\*/

Optional<Integer> optional1 = marksList.stream()

.reduce((n1,n2)->n1+n2);

System.***out***.println("Total marks: "+ optional1.get());

—--------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.util.ArrayList;

**import** java.util.Arrays;

**import** java.util.List;

**import** java.util.Optional;

/\*

\* The Stream API methods min(), max(), reduce() methods return Optional object.

\*

\* Optional is a wrapper class which wraps the result into it. This class has methods

\* to check if the stream API method has returned result or not

\*/

**public** **class** IntermediateDemo {

**public** **static** **void** main(String[] args) {

Integer[] marks = **new** Integer[] {10,20,25,16,9,48,21,57,80};

List<Integer> marksList = Arrays.*asList*(marks);

marksList.stream().map(m->m\*m).forEach(System.***out***::println);

System.***out***.println("--------------------------");

// List<Integer> squaredList = marksList.stream()

// .map(n->n\*n)

// .collect(Collectors.toList());

List<Integer> squaredList = marksList.stream()

.map(n->n\*n)

.toList();

squaredList.forEach(System.***out***::println);

System.***out***.println("-------------------------------------");

List<String> courseList= **new** ArrayList<>();

courseList.add("Java");courseList.add("java ee");

courseList.add("JavaScript");courseList.add("Spring");

courseList.add("Java");courseList.add("java ee");

//convert & collect the above course names to upper-case letter into another list and display

List<String> courseNewList = courseList.stream()

.map(s->s.toUpperCase())

.toList();

courseNewList.forEach(System.***out***::println);

System.***out***.println("----------------------------");

//display the courses with java in it

courseList.stream()

.map(s->s.toUpperCase())

.filter(s->s.contains("JAVA"))

.forEach(System.***out***::println);

System.***out***.println("Distinct courses...");

courseList.stream().distinct().forEach(System.***out***::println);

System.***out***.println("Display first 3 courses..");

courseList.stream().limit(3).forEach(System.***out***::println);

System.***out***.println("Skip first 3 courses..");

courseList.stream().skip(3).forEach(System.***out***::println);

System.***out***.println("Sorted list of courses..");

// display the courses in an order

courseList.stream().sorted().forEach(System.***out***::println);

/\*

\* Optional<Integer> java.util.stream.Stream.min(

Comparator<? super Integer> comparator

)

Comparator<Integer> : int compare(Integer, Integer)

The compare() has to be implemented using lambda,

(a,b)->a.compareTo(b)

\*/

Optional<Integer> optional = marksList.stream().min((a,b)->a.compareTo(b));

//get() extract the result from the Optional object

**if**(optional.isPresent()) {

System.***out***.println("Minimum mark: "+optional.get());

}**else** {

System.***out***.println("No min value");

}

**int** maxMark = marksList.stream().max((a,b)->a.compareTo(b)).get();

System.***out***.println("Maximum mark: "+ maxMark);

/\*

Optional<Integer> java.util.stream.Stream.reduce(

BinaryOperator<Integer> accumulator

)

\*/

Optional<Integer> optional1 = marksList.stream()

.reduce((n1,n2)->n1+n2);

System.***out***.println("Total marks: "+ optional1.get());

List<String> namesList= **new** ArrayList<>();

namesList.add("Ravi Kumar"); namesList.add("Lavanya");

namesList.add("Bhavana"); namesList.add("Vinod");

System.***out***.println("Total number of characters in all names..");

**int** totalCharacters = namesList.stream()

.map(s->s.length())

.reduce((a,b)->a+b).get();

System.***out***.println(totalCharacters);

}

}

—--------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.time.LocalDate;

**import** java.util.ArrayList;

**import** java.util.List;

**import** java.util.Map;

**import** com.wipro.model.Gender;

**import** com.wipro.model.Person;

**public** **class** PersonStreamTester {

**public** **static** **void** main(String[] args) {

List<Person> personList = **new** ArrayList<>();

*populatePersons*(personList);

System.***out***.println("List of persons...");

personList.forEach(System.***out***::println);

//get all person names

List<String> namesList = *getAllNames*(personList);

System.***out***.println("All person names in capital letters...");

namesList.forEach(System.***out***::println);

List<Person> femaleList = *getAllFemales*(personList);

System.***out***.println("Female list..");

femaleList.forEach(System.***out***::println);

//return list of persons in alphabetical order of names

List<Person> sortedPersonList = sortedPersons(personList);

System.***out***.println("Sorted list of persons..");

sortedPersonList.forEach(System.***out***::println);

/\*

\* City Name Count

\* Hyderabad 2

\* Chennai 1

\* Pune 1

\* Mumbai 1

\*/

Map<String,Integer> cityMap = getPersonCountByCity(personList);

//display cityMap

}

**private** **static** List<Person> getAllFemales(List<Person> personList) {

List<Person> femaleList = personList.stream()

.filter(p->p.getGender().equals(Gender.***FEMALE***))

.toList();

**return** femaleList;

}

**private** **static** List<String> getAllNames(List<Person> personList) {

List<String> namesList = personList.stream()

.map(p->p.getName())

.map(p->p.toUpperCase())

.toList();

**return** namesList;

}

**private** **static** **void** populatePersons(List<Person> personList) {

personList.add(**new** Person(123456789L,"Smith",Gender.***MALE***,

LocalDate.*of*(2000, 9, 12),"Hyderabad",987898879L ));

personList.add(**new** Person(213456585L,"Priya",Gender.***FEMALE***,

LocalDate.*of*(2002, 10, 15),"Pune",987898879L ));

personList.add(**new** Person(103456709L,"Clarke",Gender.***MALE***,

LocalDate.*of*(1995, 10, 21),"Mumbai",987898879L ));

personList.add(**new** Person(523450009L,"Bhavya",Gender.***FEMALE***,

LocalDate.*of*(1986, 11, 12),"Hyderabad",987898879L ));

personList.add(**new** Person(200345600L,"Virat",Gender.***MALE***,

LocalDate.*of*(2001, 12, 12),"Chennai",987898879L ));

}

}

**—----------------------------------------------------------------**

**package com.wipro.model;**

**public class Car {**

**private String manufacturerName;**

**private String carName;**

**private Double price;**

**public Car() {**

**}**

**public Car(String manufacturerName, String carName, Double price) {**

**super();**

**this.manufacturerName = manufacturerName;**

**this.carName = carName;**

**this.price = price;**

**}**

**public String getManufacturerName() {**

**return this.manufacturerName;**

**}**

**public void setManufacturerName(String manufacturerName) {**

**this.manufacturerName=manufacturerName;**

**}**

**public String getCarName() {**

**return carName;**

**}**

**public void setCarName(String carName) {**

**this.carName = carName;**

**}**

**public Double getPrice() {**

**return this.price;**

**}**

**public void setPrice(Double price) {**

**this.price = price;**

**}**

**@Override**

**public String toString() {**

**return "Car [manufacturerName=" + manufacturerName + ", carName=" + carName + ", price=" + price + "]";**

**}**

**}**

**package com.wipro.service;**

**import java.util.Comparator;**

**import java.util.List;**

**import com.wipro.model.Car;**

**// write the code using Stream API**

**public class CarService {**

**//returns sum of prices of all the cars in the list**

**public double getSumOfPrices(List<Car> carList) {**

**return carList.stream().mapToDouble(c->c.getPrice()).sum();**

**}**

**//returns list of all car names**

**public List<String> getCarNames(List<Car> carList) {**

**return carList.stream().map(c->c.getCarName()).toList();**

**}**

**//returns Cars within the given range**

**public List<Car> getCarsInRange(List<Car> carList, Double lower, Double upper) {**

**return carList.stream()**

**.filter( c -> (c.getPrice()>lower && c.getPrice() <upper))**

**.toList();**

**}**

**//returns car which has least price**

**public Car getLowestCar(List<Car> carList) {**

**return carList.stream().sorted(Comparator.*comparing*(c->c.getPrice())).toList().get(0);**

**}**

**//returns List of Cars in the descending order of price**

**public List<Car> getSortedCars(List<Car> carList){**

**return carList.stream().sorted((c1,c2)->c2.getPrice().compareTo(c1.getPrice())).toList();**

**}**

**}**

**Write a Tester class in com.wipro.app package to test the methods of CarService class.**

**package com.wipro.app;**

**import java.util.ArrayList;**

**import java.util.List;**

**import com.wipro.model.Car;**

**import com.wipro.service.CarService;**

**public class CarDemo {**

**private static List<Car> *carList*= new ArrayList<>();**

**static {**

***carList*.add(new Car("Maruti","Swift DeZire",750000.00));**

***carList*.add(new Car("Maruti","Ciaz",1250000.00));**

***carList*.add(new Car("Hyundai","Verna",1350000.00));**

***carList*.add(new Car("Mercedes Benz","EQB",7500000.00));**

***carList*.add(new Car("BMW","X1",5500000.00));**

**}**

**public static void main(String[] args) {**

**CarService service = new CarService();**

**System.*out*.println("Total price of all the cars in the showroom..");**

**System.*out*.println(service.getSumOfPrices(*carList*));**

**System.*out*.println("List Car Names..");**

**service.getCarNames(*carList*).forEach(System.*out*::println);**

**System.*out*.println("Cars in the given range..");**

**service.getCarsInRange(*carList*, 800000.00, 1500000.00).forEach(System.*out*::println);**

**System.*out*.println("Car with least price..");**

**System.*out*.println(service.getLowestCar(*carList*));**

**System.*out*.println("List of cars in descending order..");**

**service.getSortedCars(*carList*).forEach(System.*out*::println);**

**}**

**}**

**—-----------------------------------------------------------------------**

package com.wipro.app;

import java.time.LocalDate;

import java.util.ArrayList;

import java.util.Comparator;

import java.util.HashMap;

import java.util.Iterator;

import java.util.List;

import java.util.Map;

import com.wipro.model.Gender;

import com.wipro.model.Person;

public class PersonStreamTester {

public static void main(String[] args) {

List<Person> personList = new ArrayList<>();

populatePersons(personList);

System.out.println("List of persons...");

personList.forEach(System.out::println);

//get all person names

List<String> namesList = getAllNames(personList);

System.out.println("All person names in capital letters...");

namesList.forEach(System.out::println);

List<Person> femaleList = getAllFemales(personList);

System.out.println("Female list..");

femaleList.forEach(System.out::println);

//return list of persons in alphabetical order of names

List<Person> sortedPersonList = sortedPersons(personList);

System.out.println("Sorted list of persons..");

sortedPersonList.forEach(System.out::println);

/\*

\* City Name Count

\* Hyderabad 2

\* Chennai 1

\* Pune 1

\* Mumbai 1

\*/

Map<String,Integer> cityMap = getPersonCountByCity(personList);

System.out.println("City Number Of Persons");

for(Map.Entry<String,Integer> m: cityMap.entrySet()) {

System.out.println(m.getKey()+":"+m.getValue());

}

}

private static List<Person> sortedPersons(List<Person> personList) {

// return personList.stream().sorted((p1,p2)-> p1.getName().compareTo(p2.getName())).toList();

return personList.stream().sorted(Comparator.comparing(p->p.getName())).toList();

}

/\*key=Hyderabad

\* Hyderabad 3

\* Key = Pune

\* Pune 1

\* Key=Mumbai

\* Mumbai 1

\* key=Chennai

\* Chennai 1

\*/

private static Map<String, Integer> getPersonCountByCity(List<Person> personList) {

Map<String,Integer> temp = new HashMap<>();

Iterator<Person> iterator = personList.iterator();

while(iterator.hasNext()) {

String key = iterator.next().getAddress();

temp.put(key, temp.getOrDefault(key, 0)+1);

}

return temp;

}

private static List<Person> getAllFemales(List<Person> personList) {

List<Person> femaleList = personList.stream()

.filter(p->p.getGender().equals(Gender.FEMALE))

.toList();

return femaleList;

}

private static List<String> getAllNames(List<Person> personList) {

List<String> namesList = personList.stream()

.map(p->p.getName())

.map(p->p.toUpperCase())

.toList();

return namesList;

}

private static void populatePersons(List<Person> personList) {

personList.add(new Person(123456789L,"Smith",Gender.MALE,

LocalDate.of(2000, 9, 12),"Hyderabad",987898879L ));

personList.add(new Person(213456585L,"Priya",Gender.FEMALE,

LocalDate.of(2002, 10, 15),"Pune",987898879L ));

personList.add(new Person(103456709L,"Clarke",Gender.MALE,

LocalDate.of(1995, 10, 21),"Mumbai",987898879L ));

personList.add(new Person(523450009L,"Bhavya",Gender.FEMALE,

LocalDate.of(1986, 11, 12),"Hyderabad",987898879L ));

personList.add(new Person(200345600L,"Virat",Gender.MALE,

LocalDate.of(2001, 12, 12),"Chennai",987898879L ));

personList.add(new Person(150345600L,"Rishab",Gender.MALE,

LocalDate.of(1999, 10, 20),"Hyderabad",787898800L ));

}

}

**—-----------------------------------------------------------------------------**

**SOLID Principles & Design Patterns**

**SOLID** principles:

**S**ingle State of Responsibility

**O**pen for extension, Closed for Modification

**L**iskov Substitution

**I**nterface Segregation

**D**ependency Inversion

1. Single state of responsibility

A well designed class should contains methods that related to the properties of the class. I.e the behavior of an entity should reflect the nature of the entity and should not any behavior that is beyond its nature.

For. A class Employee behaviour should only be about the employee. For example, a method that returns total pay of an employee, which dept employee works. Etc but should not methods that are indirectly related for ex. Details of the department in which he/she is working.

**A well designed POJO class follows single-state of responsibility.**

POJO : Plain Old Java Object class

**A POJO class should have foll. Features:**

1. Data members have to be private
2. There should be zero-arg constructor(default constructor)
3. Overloaded constructor methods are optional.
4. Every data member should have a pair of getter/setter methods which have to be public.
5. Methods that return String representation of an object i.e overriding toString() method is optional.
6. Override equals() & hashCode() methods which is optional.

**Open for extension, closed for modification.**

Once a class is **defined, tested and compiled**, It should not be altered because there might many applications which are relying on the compiled class.

What if we want to upgrade the class to add new features or behaviour?

**Create a subclass** of the existing class so that the original class is not disturbed.

This is called an extension.

**Liskov Substitution**

The subclass or child class should complement its superclass i.e parent class but not substitute the parent class.

Square extends Rectangle

ColoredRectangle extends Rectangle

ColoredRectangle is not substituting the behaviour of Rectangle whereas Square is substituting the behaviour of Rectangle.

This is not recommended.

Solution:

**Shape**

**Rectangle Square Circle**

**Interface Segregation**

public interface **I**

m1();

m2();

m3();

}

public abstract class A implements I {

@Override

public void m1(){

}

}

Note: We cannot instantiate class A. What if I want to instantiate class A?

public class B implements I {

}

public class C implements I {

}

**GO for Interface Segregation**

Instead of defining an interface with more than one abstract method,

Define only functional interface so that classes will decide which interfaces to implement.

@FunctionlInterface

public interface **I1**

m1();

}

@FunctionlInterface

public interface **I2**

m2();

}

@FunctionlInterface

public interface @FunctionlInterface

public interface **I**

m3();

}

public class A implements I1{

//override only m1() method

}

public class B implements I2{

//override only m2() method

}

public class B implements I3{

//override only m3() method

}

public class D implements I1, I2{

//override m1() and m2() method

}

**Dependency Inversion**

**Object composition is has-a relationship**

Dependent classes and Dependency classes

Generally A class is composition of other classes.

Ex. Car has an Engine, has Tyre(s), has Navigation etc.

Here Car is called dependent class and the classes, Engine, Tyre etc are called as dependency classes.

**How to inject dependencies into dependent object?**

1. **constructor injection: inject dependencies into dependent object through constructor method.**
2. **setter injection: inject dependencies into dependent object through setter methods.**

**Ex.**

***//dependency classes***

**public class Engine{}**

**public class Tyre{}**

***//dependent class***

**public class Car{**

**private Engine engine;**

**Tyre [] tyres;**

**}**

**App**

**—-------------------------------------------------**

***//dependency objects***

**Engine marutiEngine = new Engine();**

**Tyre[] tyres= new Tyre[4];**

**// Injecting the above dependency into dependent class,**

**//constructor injection**

**Car myCar = new Car(marutiEngine, tyres);**

**//setter injection**

**Car myCarNew = new Car();**

**myCarNew.setEngine(marutiEngine);**

**myCarNew.setTryres(tyres);**

**—----------------------------------------------------------------------------------------------------------------**

**Java Design Patterns**

A design pattern is a well-described solution to a common software problem.

Some of the benefits of using design patterns are:

1. Design patterns are already defined and provide an *industry-standard approach* to solving a recurring problem, so it saves time if we sensibly use the design pattern. There are many Java design patterns that we can use in our Java-based projects.
2. Using design patterns promotes *reusability* that leads to more *robust* and highly maintainable code. It helps in reducing the total cost of ownership (TCO) of the software product.
3. Since design patterns are already defined, it makes our code easy to understand and debug. It leads to faster development and new members of the team understand it easily.

Java design patterns are divided into three categories -

1. **Creational,**
2. **Structural, and**
3. **Behavioral design patterns.**
4. **Creational pattern**

**Java design pattern for creating the objects.**

**A. Singleton Design Pattern**

**B. Factory Design Pattern**

**B. Builder design Pattern**

1. **Singleton Design pattern**

This pattern describes how to create a single instance of a class.

By default, a class can have any number of instances. Singleton design pattern is implementation of a class that enable to create only one instance.

**package** com.wipro.model;

**public** **class** Singleton {

**private** **static** Singleton *singleton*=**null**;

**private** Singleton() {

}

**public** **static** Singleton getInstance() {

**if**(*singleton*==**null**) {

*singleton* = **new** Singleton();

}

**return** *singleton*;

}

}

**package** com.wipro.app;

**import** com.wipro.model.Singleton;

**public** **class** SingletonDemo {

**public** **static** **void** main(String[] args) {

Singleton singleton= Singleton.*getInstance*();

System.***out***.println(singleton);

Singleton singletonNew = Singleton.*getInstance*();

System.***out***.println(singletonNew);

}

}

1. **Factory Design Pattern**

The factory design pattern is used when we have a superclass with multiple subclasses and based on input, we need to return one of the subclasses.

This pattern takes out the responsibility of the instantiation of a Class from the client program to the factory class.

**package** com.wipro.model;

**public** **enum** CarTyp {

***SEDAN***, ***SUV***, ***SMALL***

}

—---------------------------------------------------------------------------------------------------------------

**package** com.wipro.model;

**public** **class** Car {

**private** CarTyp carTyp;

**private** String manufacturer;

**private** String model;

**private** Double price;

**public** Car() {

}

**public** Car(String manufacturer, String model, Double price) {

**super**();

**this**.manufacturer = manufacturer;

**this**.model = model;

**this**.price = price;

}

**public** CarTyp getCarTyp() {

**return** carTyp;

}

**public** **void** setCarTyp(CarTyp carTyp) {

**this**.carTyp = carTyp;

}

**public** String getManufacturer() {

**return** manufacturer;

}

**public** **void** setManufacturer(String manufacturer) {

**this**.manufacturer = manufacturer;

}

**public** String getModel() {

**return** model;

}

**public** **void** setModel(String model) {

**this**.model = model;

}

**public** Double getPrice() {

**return** price;

}

**public** **void** setPrice(Double price) {

**this**.price = price;

}

}

—---------------------------------------------------------------------------------------------------------------

**package** com.wipro.model;

**public** **class** SmallCar **extends** Car{

**public** SmallCar() {

}

**public** SmallCar(String manufacturer, String model, Double price) {

**super**(manufacturer, model, price);

}

}

—-------------------------------------------------------------------------------------

**package** com.wipro.model;

**public** **class** SedanCar **extends** Car{

**public** SedanCar() {

}

**public** SedanCar(String manufacturer, String model, Double price) {

**super**(manufacturer, model, price);

}

}

—-------------------------------------------------------------------------------------------------

**package** com.wipro.model;

**public** **class** SuvCar **extends** Car{

**public** SuvCar() {

}

**public** SuvCar(String manufacturer, String model, Double price) {

**super**(manufacturer, model, price);

}

}

—----------------------------------------------------------------------------------------------------------------------

**package** com.wipro.model;

/\*

\* Factory creating different types of Cars

\*/

**public** **class** CarFactory {

**public** **static** Car createCar(CarTyp carTyp, String manufacturer,

String model, Double price) {

**switch**(carTyp) {

**case** CarTyp.***SMALL***:

**return** **new** SmallCar(manufacturer,model,price);

**case** CarTyp.***SEDAN***:

**return** **new** SedanCar(manufacturer,model,price);

**case** CarTyp.***SUV***:

**return** **new** SuvCar(manufacturer,model,price);

}

**return** **null**;

}

}

—-------------------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** com.wipro.model.Car;

**import** com.wipro.model.CarFactory;

**import** com.wipro.model.CarTyp;

**public** **class** CarFactoryDemo {

**public** **static** **void** main(String[] args) {

// String carType = "sedan";

String carType = "suv";

CarTyp carTyp;

**if**(carType.equalsIgnoreCase("SEDAN")) {

carTyp = CarTyp.***SEDAN***;

}**else** **if**(carType.equalsIgnoreCase("small")) {

carTyp = CarTyp.***SMALL***;

}**else** {

carTyp = CarTyp.***SUV***;

}

// Car myCar = CarFactory.createCar(carTyp, "Skoda", "Rapid", 1565000.00);

Car myCar = CarFactory.*createCar*(carTyp, "BMW", "X1", 5650000.00);

System.***out***.println(myCar.getManufacturer()+","+myCar.getModel()+","+myCar.getPrice());

};

}

—--------------------------------------------------------------------------------------------------------------------

1. **Builder Design Pattern**

**This is a creational design pattern that enables us to construct complex objects step by step.**

**This pattern useful when a class has many data members since creating through constructor will be too long.**

**White designing Builder pattern, we need to go for nested static class.**

**package** com.wipro.model;

**public** **class** Employee {

**private** Integer empid;

**private** String name;

**private** String departmentName;

**public** Employee(EmployeeBuilder builder) {

empid = builder.empid;

name = builder.name;

departmentName= builder.departmentName;

}

**public** **static** **class** EmployeeBuilder{

**private** Integer empid;

**private** String name;

**private** String departmentName;

**public** EmployeeBuilder(Integer empid) {

**this**.empid= empid;

}

**public** EmployeeBuilder getName(String name) {

**this**.name = name;

**return** **this**;

}

**public** EmployeeBuilder getDepartmentName(String departmentName) {

**this**.departmentName=departmentName;

**return** **this**;

}

**public** Employee build() {

**return** **new** Employee(**this**);

}

}

**public** Integer getEmpid() {

**return** empid;

}

**public** **void** setEmpid(Integer empid) {

**this**.empid = empid;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getDepartmentName() {

**return** departmentName;

}

**public** **void** setDepartmentName(String departmentName) {

**this**.departmentName = departmentName;

}

}

**—-------------------------------------------------------------------------------------------------**

package com.wipro.app;

import com.wipro.model.Employee;

public class EmployeeBuilderDemo {

public static void main(String[] args) {

Employee employee =

new Employee.EmployeeBuilder(1001)

.getName("Smith")

.getDepartmentName("Accounts")

.build();

System.*out*.println(employee.getName());

}

}

**—------------------------------------------------------------------------------------------------**

**Structural Design Patterns**

Structural patterns are concerned with how classes and objects are composed to form larger structures..

**Important structural design patterns are**

• **Adapter**

• Bridge

• Composite

• Decorator:

A Decorator pattern can be used to attach additional responsibilities to an object

either statically or dynamically.

Ex. BufferedInputStream, DataInputStream

• Facade

• Flyweight

• Proxy

Provide a surrogate or placeholder for another object to control access to it.

The definition itself is very clear and proxy design pattern is used when we want to provide controlled access of a functionality.

Proxy means ‘in place of’, representing’ or ‘in place of’ or ‘on behalf of’ are literal meanings of proxy and that directly explains Proxy Design Pattern.

**Adapter pattern**

An Adapter pattern acts as a connector between two incompatible interfaces that otherwise cannot be connected directly.

An Adapter wraps an existing class with a new interface so that it becomes compatible with theclient’s interface.

<https://refactoring.guru/design-patterns/adapter/java/example>

**package** com.wipro.model;

**public** **interface** Movable {

//returns speed in MPH

**double** getSpeed();

}

**package** com.wipro.model;

**public** **class** BugattiVeyron **implements** Movable {

@Override

**public** **double** getSpeed() {

**return** 268;

}

}

**package** com.wipro.model;

**public** **interface** MovableAdapter {

// returns speed in KM/H

**double** getSpeed();

}

**package** com.wipro.model;

**public** **class** MovableAdapterImpl **implements** MovableAdapter {

**private** Movable luxuryCars;

**public** MovableAdapterImpl() {

}

**public** MovableAdapterImpl(Movable luxuryCars) {

**this**.luxuryCars = luxuryCars;

}

@Override

**public** **double** getSpeed() {

**return** luxuryCars.getSpeed() \* 1.6;

}

}

**package** com.wipro.app;

**import** com.wipro.model.BugattiVeyron;

**import** com.wipro.model.Movable;

**import** com.wipro.model.MovableAdapter;

**import** com.wipro.model.MovableAdapterImpl;

**public** **class** AdapterDemo {

**public** **static** **void** main(String[] args) {

Movable bugatti = **new** BugattiVeyron();

System.***out***.println("Speed of Bugatti in MPH is "+ bugatti.getSpeed());

MovableAdapter movableAdapter = **new** MovableAdapterImpl(bugatti);

System.***out***.println("Speed in KMPH: "+ movableAdapter.getSpeed());

}

}

—-----------------------------------------------------------------------------------------

**Behavioural Design Pattern**

Behavioural patterns are concerned with algorithms and the **assignment of responsibilities between objects.**

Behavioural patterns describe not just patterns of objects or classes but also the patterns of communication between them.

**Important behavioural patterns are**

**• Chain of responsibility :**

This design pattern is applied in Java EE Filters

The request coming from the client(browser) is received by one filter which performs some task and hands over to another filter which again performs some other tasks and hands over to next filter and this process continues until the request object reaches the Servlet.

• **Iterator**

The Iterator interface of Collection frameworks takes the responsibility of iterating through Collection implementations such as ArrayList.

ArrayList object returns Iterator object which in turn traverses through the ArrayList.

The responsibility of holding of objects is taken care by ArrayList and the responsibility of traversing through the ArrayList is taken care by Iterator.

List<Integer> marksList = new ArrayList<>();

marksList.add(90);

Iterator<Integer> iterator = marksList.iterator();

while(iterator.hasNext()){

}

• Mediator

• Memento

**• Observer**

In Java EE, we have Listeners whose job is to listen the events that take place such as adding, removing, modifying attribute of Request object, Session object, Application Context objects.

—-----------------------------------------------------------------------------------

**Java IO**

**package com.wipro.app;**

**import java.util.Map;**

**import java.util.Properties;**

**import java.util.Scanner;**

**/\***

**\* Properties is a subclass of Hashtable where data is stored in form of**

**\* key:value pairs. These are legacy classes retrofitted into Map interface.**

**\* System.in, System.out & System.err are predefined streams**

**\*/**

**public class SystemPropertiesDemo {**

**public static void main(String[] args) {**

**Properties properties = System.*getProperties*();**

**for(Map.Entry<Object,Object> m : properties.entrySet()) {**

**System.*out*.println(m.getKey()+":"+m.getValue());**

**}**

**System.*out*.println("---------------------------");**

**String value = System.*getProperty*("java.vm.version");**

**System.*out*.println("Java Version: "+ value);**

**Scanner scanner = new Scanner(System.*in*);**

**System.*out*.println("Enter your name: ");**

**String name = scanner.nextLine();**

**System.*out*.println(name);**

**System.*err*.println(name);**

**}**

**}**

**—-----------------------------------------------------------------------------**

**package com.wipro.app;**

**import java.io.BufferedOutputStream;**

**import java.io.BufferedReader;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**/\*\***

**\* System.in is the KBD which is connected to InputStreamObject which in turn is**

**\* connected to BufferedReader**

**\***

**\* KBD -> InputStreamReader->BufferedReader -> program**

**\*/**

**public class StudentNamesFile {**

**public static void main(String[] args) {**

**try(**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));**

**/\*If FileOutputStream has no second argument then it is opened in write mode.**

**\* If second argument is set to true, then file is opened in append mode**

**\***

**\*/**

**BufferedOutputStream bos = new BufferedOutputStream(**

**new FileOutputStream("/home/rps/Documents/data/names.txt",true));**

**){**

**System.*out*.println("Enter names of the students(cntrl-z to stop): ");**

**int ch;**

**while( (ch = br.read()) != -1) {**

**bos.write(ch);**

**}**

**br.close();**

**System.*out*.println("File created..");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**—-------------------------------------------------------**

**package com.wipro.app;**

**import java.io.BufferedInputStream;**

**import java.io.BufferedOutputStream;**

**import java.io.BufferedReader;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**/\*\***

**\* System.in is the KBD which is connected to InputStreamObject which in turn is**

**\* connected to BufferedReader**

**\***

**\* KBD -> InputStreamReader->BufferedReader -> program**

**\*/**

**public class StudentNamesFile {**

**public static void main(String[] args) {**

**try(**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));**

**/\*If FileOutputStream has no second argument then it is opened in write mode.**

**\* If second argument is set to true, then file is opened in append mode**

**\***

**\*/**

**BufferedOutputStream bos = new BufferedOutputStream(**

**new FileOutputStream("/home/rps/Documents/data/names.txt",true));**

**){**

**System.*out*.println("Enter names of the students(cntrl-d to stop): ");**

**int ch;**

**while( (ch = br.read()) != -1) {**

**bos.write(ch);**

**}**

**br.close();**

**System.*out*.println("File created..");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**try(**

**BufferedInputStream bif =**

**new BufferedInputStream(new FileInputStream("/home/rps/Documents/data/names.txt"))**

**){**

**int ch;**

**while( (ch = bif.read()) != -1) {**

**System.*out*.print((char)ch);**

**}**

**bif.close();**

**System.*out*.println("End of file");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**—--------------------------------------------------------------------------**

**package com.wipro.app;**

**import java.io.BufferedInputStream;**

**import java.io.BufferedOutputStream;**

**import java.io.BufferedReader;**

**import java.io.File;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**/\*\***

**\* System.in is the KBD which is connected to InputStreamObject which in turn is**

**\* connected to BufferedReader**

**\***

**\* KBD -> InputStreamReader->BufferedReader -> program**

**\*/**

**public class StudentNamesFile {**

**public static void main(String[] args) {**

**File file = new File("/home/rps/Documents/data/names.txt");**

**try(**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));**

**/\*If FileOutputStream has no second argument then it is opened in write mode.**

**\* If second argument is set to true, then file is opened in append mode**

**\***

**\*/**

**BufferedOutputStream bos = new BufferedOutputStream(**

**new FileOutputStream(file,true));**

**){**

**System.*out*.println("Enter names of the students(cntrl-d to stop): ");**

**int ch;**

**while( (ch = br.read()) != -1) {**

**bos.write(ch);**

**}**

**br.close();**

**System.*out*.println("File created..");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**if(file.exists() && file.canRead()) {**

**try(**

**BufferedInputStream bif =**

**new BufferedInputStream(new FileInputStream(file))**

**){**

**int ch;**

**while( (ch = bif.read()) != -1) {**

**System.*out*.print((char)ch);**

**}**

**bif.close();**

**System.*out*.println("End of file");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**}else {**

**System.*out*.println("Unable to open the file");**

**}**

**}**

**}**

**—--------------------------------------------------------------**

**package com.wipro.app;**

**import java.io.BufferedInputStream;**

**import java.io.BufferedOutputStream;**

**import java.io.BufferedReader;**

**import java.io.File;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**/\*\***

**\* System.in is the KBD which is connected to InputStreamObject which in turn is**

**\* connected to BufferedReader**

**\***

**\* KBD -> InputStreamReader->BufferedReader -> program**

**\*/**

**public class StudentNamesFile {**

**public static void main(String[] args) {**

**File file = new File("/home/rps/Documents/data/names.txt");**

**try(**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));**

**/\*If FileOutputStream has no second argument then it is opened in write mode.**

**\* If second argument is set to true, then file is opened in append mode**

**\***

**\*/**

**BufferedOutputStream bos = new BufferedOutputStream(**

**new FileOutputStream(file,true));**

**){**

**System.*out*.println("Enter names of the students(cntrl-d to stop): ");**

**int ch;**

**while( (ch = br.read()) != -1) {**

**bos.write(ch);**

**}**

**br.close();**

**System.*out*.println("File created..");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**if(file.exists() && file.canRead()) {**

**try(**

**// BufferedInputStream bif =**

**// new BufferedInputStream(new FileInputStream(file))**

**FileInputStream fis = new FileInputStream(file)**

**){**

**int ch;**

**// while( (ch = bif.read()) != -1) {**

**// System.out.print((char)ch);**

**// }**

**// bif.close();**

**while( (ch = fis.read()) != -1) {**

**System.*out*.print((char)ch);**

**}**

**fis.close();**

**System.*out*.println("End of file");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**}else {**

**System.*out*.println("Unable to open the file");**

**}**

**}**

**}**

**—------------------------------------------------------------------**

**package com.wipro.app;**

**import java.io.BufferedInputStream;**

**import java.io.BufferedOutputStream;**

**import java.io.BufferedReader;**

**import java.io.DataInputStream;**

**import java.io.DataOutputStream;**

**import java.io.File;**

**import java.io.FileInputStream;**

**import java.io.FileOutputStream;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**/\***

**\* To read/write primitive-type data values from/to files, use DataInputStream**

**\* and DataOutputStream classes respectively**

**\*/**

**public class DataInputOutputStreamDemo {**

**public static void main(String[] args) {**

**File file = new File("/home/rps/Documents/data/student-details.txt");**

**try(**

**/\***

**\* reading data from KBD through BufferedReader object and then**

**\* writing the data into file through DataOutputStream object**

**\*/**

**DataOutputStream dos =**

**new DataOutputStream(new BufferedOutputStream( new FileOutputStream(file)));**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));**

**){**

**System.*out*.println("Enter student name: ");**

**String name= br.readLine();**

**System.*out*.println("Eneter age in years: ");**

**int age = Integer.*parseInt*(br.readLine());**

**System.*out*.println("Enter mobile number: ");**

**long mobile = Long.*parseLong*(br.readLine());**

**dos.writeUTF(name);**

**dos.write(age);**

**dos.writeLong(mobile);**

**System.*out*.println("File created");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**if(file.exists() && file.canRead()) {**

**try(**

**DataInputStream dis = new DataInputStream**

**(new BufferedInputStream**

**(new FileInputStream(file)))**

**){**

**String name = dis.readUTF();**

**int age = dis.read();**

**long mobile = dis.readLong();**

**System.*out*.println(name+","+age+","+mobile);**

**System.*out*.println("End of file");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**}else {**

**System.*out*.println("Unable to open the file");**

**}**

**}**

**}**

**—-----------------------------------------------------------------**

**Character Streams**

Top level classes:

Reader and Writer classes

The Java platform stores character values using Unicode conventions. Character stream I/O automatically translates this internal format to and from the local character set.

**When to use Character Stream over Byte Stream?**

In Java, characters are stored using Unicode conventions. **Character stream is useful when we want to process text files**. These text files can be processed character by character. Character size is typically 16 bits.

**When to use Byte Stream over Character Stream?**

Byte oriented reads byte by byte. **A byte stream is suitable for processing raw data like binary files such as audio/video/pictures.**

**Reader class:**

**BufferedReader, FIleReader, InputStreamReader**

**Writer class:**

**BufferedWriter, FileWriter, PrintWriter, OutputStreamWriter**

**package com.wipro.app;**

**import java.io.BufferedReader;**

**import java.io.BufferedWriter;**

**import java.io.File;**

**import java.io.FileReader;**

**import java.io.FileWriter;**

**import java.io.IOException;**

**import java.io.InputStreamReader;**

**import java.io.PrintWriter;**

**public class FileReaderWriterDemo {**

**public static void main(String[] args) {**

**File file = new File("/home/rps/Documents/data/story.txt");**

**try(**

**BufferedReader br = new BufferedReader(new InputStreamReader(System.*in*));**

**PrintWriter pw = new PrintWriter(**

**new BufferedWriter(**

**new FileWriter(file,true)));**

**){**

**System.*out*.println("Enter text (cntrl-d to stop): \n");**

**int ch;**

**while(( ch =br.read()) != -1) {**

**pw.print((char)ch);**

**}**

**System.*out*.println("File created");**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**if(file.exists() && file.canRead()) {**

**try(**

**BufferedReader br = new BufferedReader(**

**new FileReader(file)**

**);**

**){**

**int ch;**

**while( (ch = br.read()) != -1) {**

**System.*out*.print((char)ch);**

**}**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**}else {**

**System.*out*.println("Unable to open the file");**

**}**

**}**

**}**

Write a program to copy the contents of story.txt file into javanotes.txt file.

Hint: Open **story.txt** in read mode using FileReader class and **javanotes.txt** in write mode using FileWriter class.

**package** com.wipro.app;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileReader;

**import** java.io.FileWriter;

**import** java.io.IOException;

**public** **class** CopyFileDemo {

**public** **static** **void** main(String[] args) {

File sourceFile= **new** File("/home/rps/Documents/data/story.txt");

File destFile=**null**;

**if**(sourceFile.exists() && sourceFile.canRead()) {

destFile = **new** File("/home/rps/Documents/data/javanotes.txt");

**try**(

BufferedReader br = **new** BufferedReader(**new** FileReader(sourceFile));

BufferedWriter bw = **new** BufferedWriter(**new** FileWriter(destFile));

){

**int** ch;

**while**( (ch= br.read() ) != -1 ) {

bw.write((**char**)ch);

}

System.***out***.println("File copied..");

}**catch**(IOException e) {

e.printStackTrace();

}

}**else** {

System.***out***.println("Unable to open the source file");

}

System.***out***.println("Contents of destination file: ");

**try**(

BufferedReader br = **new** BufferedReader(**new** FileReader(destFile))

){

**int** ch;

**while**( (ch=br.read()) != -1) {

System.***out***.print((**char**)ch);

}

}**catch**(IOException e) {

e.printStackTrace();

}

}

}

—-------------------------------------------------------------------------------------------

**Object Serialization**

The process of storing objects of a class that implements Serializable interface into permanent storage device is called Object Serialization

Serializable objects can be converted into stream of bytes

This stream of bytes can be written into a file, which is done by **ObjectOutputStream** class methods.

These bytes can be read back to re-create the objects, which is done by **ObjectInputStream** class methods. This process is called as Deserialization.

Only those objects that implement **java.io.Serializable** can be persisted into a file.

Note: **Serializable** is a marker interface i.e an interface without any abstract methods.

If a class implements Serializable interface, it is an indication to JVM to prepare serialising objects into file.

**serialVersionUid** is a unique long value generated by JVM at runtime which will be stored in the file along with the objects. This id is generated only to a class that implements Serializable interface.

**File** is managed by OS where data is stored as a sequence of bytes. These files are also called as flat files. Manipulation of files is slower.

**Table** is database object managed by RDBMS platforms such as Oracle, MySQL, PostgreSQL etc.

RDBM provides a layer of abstraction between the programs and the OS file system which provides logical objects such as Tables, Views, Indexes etc.

We always connect to the abstract layer by applying SQL commands such as Create, Alter, Drop, Insert, Delete, Update, Select etc.

It is convenient and efficient to handle data when it is stored in RDBMS data stores.

To handle data which is in RDBMS data store, Java provides as API with which you can access the RDBMS which is called **JDBC API.**

**package com.wipro.model;**

**public class Employee implements Serializable{**

**private Integer empid;**

**private String name;**

**private String job;**

**private Double salary;**

**private String deptName;**

**public Employee() {**

**}**

**public Employee(Integer empid, String name, String job, Double salary, String deptName) {**

**super();**

**this.empid = empid;**

**this.name = name;**

**this.job = job;**

**this.salary = salary;**

**this.deptName = deptName;**

**}**

**public Integer getEmpid() {**

**return empid;**

**}**

**public void setEmpid(Integer empid) {**

**this.empid = empid;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public String getJob() {**

**return job;**

**}**

**public void setJob(String job) {**

**this.job = job;**

**}**

**public Double getSalary() {**

**return salary;**

**}**

**public void setSalary(Double salary) {**

**this.salary = salary;**

**}**

**public String getDeptName() {**

**return deptName;**

**}**

**public void setDeptName(String deptName) {**

**this.deptName = deptName;**

**}**

**@Override**

**public String toString() {**

**return "Employee [empid=" + empid + ", name=" + name + ", job=" + job + ", salary=" + salary + ", deptName="**

**+ deptName + "]";**

**}**

**}**

**package** com.wipro.app;

**import** java.io.BufferedReader;

**import** java.io.File;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.io.InputStreamReader;

**import** java.io.ObjectOutputStream;

**import** com.wipro.model.Employee;

**public** **class** EmployeeWriter {

**private** **static** BufferedReader *bufferedReader* =

**new** BufferedReader(**new** InputStreamReader(System.***in***));

**public** **static** **void** main(String[] args) {

File file = **new** File("/home/rps/Documents/data/employee.ser");

**try**(

ObjectOutputStream oos = **new** ObjectOutputStream(

**new** FileOutputStream(file));

){

**while**(**true**) {

Employee employee = **new** Employee();

*acceptEmployee*(employee);

oos.writeObject(employee);

System.***out***.println("To stop, press x: ");

**if**(*bufferedReader*.readLine().equalsIgnoreCase("x")) **break**;

}

System.***out***.println("Employees saved to file");

}**catch**(IOException e) {

e.printStackTrace();

}

}

**private** **static** **void** acceptEmployee(Employee employee) {

**try** {

System.***out***.println("Enter employee Id: ");

employee.setEmpid(Integer.*parseInt*(*bufferedReader*.readLine()));

System.***out***.println("Enter employee name: ");

employee.setName(*bufferedReader*.readLine());

System.***out***.println("Enter employee job: ");

employee.setJob(*bufferedReader*.readLine());

System.***out***.println("Enter salary: ");

employee.setSalary(Double.*parseDouble*(*bufferedReader*.readLine()));

System.***out***.println("Enter department name: ");

employee.setDeptName(*bufferedReader*.readLine());

}**catch**(IOException e) {

e.printStackTrace();

}

}

}

**package** com.wipro.app;

**import** java.io.EOFException;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.IOException;

**import** java.io.ObjectInputStream;

**import** com.wipro.model.Employee;

/\*

\* display contents of file, employee.ser

\*/

**public** **class** EmployeeReader {

**public** **static** **void** main(String[] args) {

File file = **new** File("/home/rps/Documents/data/employee.ser");

**if**(file.exists() && file.canRead()) {

**try**(

ObjectInputStream ois = **new** ObjectInputStream(**new** FileInputStream(file));

){

Employee employee;

**while**( ( employee = (Employee) ois.readObject() )!= **null**) {

System.***out***.println(employee);

}

}**catch**(EOFException e) {

System.***out***.println("End of file");

}**catch**(IOException e) {

e.printStackTrace();

}**catch**(ClassNotFoundException e) {

e.printStackTrace();

}

}

}

}

—---------------------------------------------------------------------

**package** com.wipro.model;

**import** java.io.Serializable;

**import** java.time.LocalDate;

/\*

\* transient fields will not be serialized

\*/

**public** **class** Employee **implements** Serializable{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**private** Integer empid;

**private** String name;

**private** String job;

**transient** **private** Double salary;

**private** String deptName;

**public** Employee() {

}

**public** Employee(Integer empid, String name, String job, Double salary, String deptName) {

**super**();

**this**.empid = empid;

**this**.name = name;

**this**.job = job;

**this**.salary = salary;

**this**.deptName = deptName;

}

**public** Integer getEmpid() {

**return** empid;

}

**public** **void** setEmpid(Integer empid) {

**this**.empid = empid;

}

**public** String getName() {

**return** name;

}

**public** **void** setName(String name) {

**this**.name = name;

}

**public** String getJob() {

**return** job;

}

**public** **void** setJob(String job) {

**this**.job = job;

}

**public** Double getSalary() {

**return** salary;

}

**public** **void** setSalary(Double salary) {

**this**.salary = salary;

}

**public** String getDeptName() {

**return** deptName;

}

**public** **void** setDeptName(String deptName) {

**this**.deptName = deptName;

}

@Override

**public** String toString() {

**return** "Employee [empid=" + empid + ", name=" + name + ", job=" + job + ", salary=" + salary + ", deptName="

+ deptName + "]";

}

}

—----------------------------------------------------------------------------------------------------

**Java NIO**

**NIO stands for Non-Blocking IO**

**The key features were:**

**•Channels and Selectors:** A channel is an abstraction on lower-level file system features, e.g. memory-mapped files.

**•Buffers:** Buffering for all primitive classes

**•Charset:** Charset (java.nio.charset), encoders, and decoders to map bytes and Unicode symbols

**Path**

A Path object is a hierarchical representation of the path on a system to the file or directory.

The java.nio.file.Path interface is the primary entry point for working with the NIO 2 API

Path path = Paths.get(“pathaname”);

—-------------------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.io.BufferedWriter;

**import** java.io.IOException;

**import** java.nio.charset.Charset;

**import** java.nio.file.Files;

**import** java.nio.file.Path;

**import** java.nio.file.Paths;

**public** **class** PathDemo {

**public** **static** **void** main(String[] args) {

Path path = Paths.*get*("/home/rps/Documents/data/shakespere.txt");

**try**(

BufferedWriter writer =

Files.*newBufferedWriter*(path, Charset.*forName*("UTF-8"))

){

System.***out***.println(path.getFileName());//shakespere.txt

writer.write("To be, or not to be. That is the question.");

}**catch**(IOException e){

e.printStackTrace();

}

}

}

—--------------------------------------------------------------------------------------------

**package** com.wipro.app;

**import** java.io.IOException;

**import** java.io.OutputStream;

**import** java.nio.file.Files;

**import** java.nio.file.Path;

**import** java.nio.file.Paths;

/\*

\* copy contents of existing file into a new file

\*/

**public** **class** NIOCopyDemo {

**public** **static** **void** main(String[] args) {

Path sourcePath = Paths.*get*("/home/rps/Documents/data/shakespere.txt");

Path destPath = Paths.*get*("/home/rps/Documents/data/shakespere-new.txt");

**try**(

OutputStream os = Files.*newOutputStream*(destPath)

){

Files.*copy*(sourcePath, os);

System.***out***.println("File copied..");

}**catch**(IOException e) {

e.printStackTrace();

}

}

}

—-----------------------------------------------------------------------------------------------------

Java Non-blocking Input Output (NIO) consist of the following core components:

• **Channels**

**• Buffers**

**• Selectors**

• **Pipe** and **FileLock** are utility classes that are used in conjunction with the above three core components.

The class methods of java.io package are **blocking IO** where the class methods of java.nio package are **non-blocking IO.**

Blocking IO is synchronous programming whereas blocking IO comes under asynchronous programming.

In blocking IO, when a method is called to access data, say remote data, the program waits for the outcome of the method. I.e next statement will not get executed until the method that made a remote call returns the results.

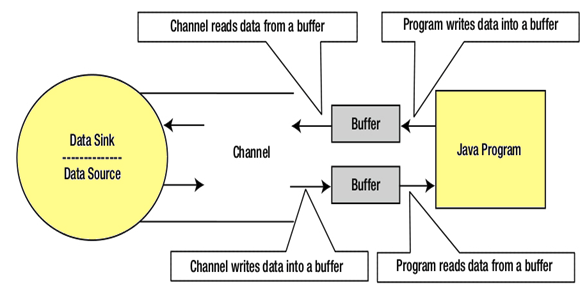
In non-blocking IO, when an asynchronous method makes a call to fetch data from remote place, it will promise to return the result at some point of time meanwhile allowing execution of the next statement.

Java NIO Channels are similar to streams with a few differences:

•You can both read and write to a Channels. Streams are typically one-way (read or write).

•Channels can be read and written asynchronously.

•Channels always read to, or write from, a Buffer



**Channel Implementations**

Here are the most important Channel implementations in Java NIO:

•FileChannel

•DatagramChannel

•SocketChannel

•ServerSocketChannel

The FileChannel reads data from and to files.

The DatagramChannel can read and write data over the network via UDP.

The SocketChannel can read and write data over the network via TCP.

The ServerSocketChannel allows you to listen for incoming TCP connections, like a web server does. For each incoming connection a SocketChannel is created.

**Buffer**

A buffer is essentially a block of memory into which you can write data, which you can then later read again.

This memory block is wrapped in a NIO Buffer object, which provides a set of methods that makes it easier to work with the memory block.

**Basic Buffer Usag**e

Using a Buffer to read and write data typically follows this little 4-step process:

1.Write data into the Buffer

2.Call buffer.flip()

3.Read data out of the Buffer

4.Call buffer.clear() or buffer.compact()

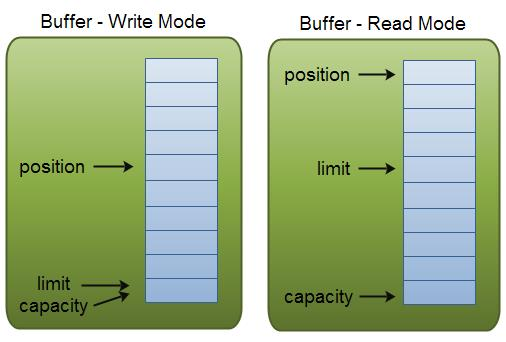
**A Buffer has three properties you need to be familiar with, in order to understand how a Buffer works.**

**These are:**

**•capacity**

**•position**

**•limit**

****

[**https://jenkov.com/tutorials/java-nio/buffers.html**](https://jenkov.com/tutorials/java-nio/buffers.html)

## **Writing Data to a Buffer**

You can write data into a Buffer in two ways:

1. Write data from a Channel into a Buffer
2. Write data into the Buffer yourself, via the buffer's put() methods.

Here is an example showing how a Channel can write data into a Buffer:

int bytesRead = inChannel.read(buf); //read into buffer.

Here is an example that writes data into a Buffer via the put() method:

buf.put(127);

There are many other versions of the put() method, allowing you to write data into the Buffer in many different ways. For instance, writing at specific positions, or writing an array of bytes into the buffer. See the JavaDoc for the concrete buffer implementation for more details.

## **flip()**

The flip() method switches a Buffer from **writing mode to reading mode**. Calling flip() sets the position back to 0, and sets the limit to where position just was.

In other words, position now marks the reading position, and limit marks how many bytes, chars etc. were written into the buffer - the limit of how many bytes, chars etc. that can be read.

## **Reading Data from a Buffer**

There are two ways you can read data from a Buffer.

1. Read data from the buffer into a channel.
2. Read data from the buffer yourself, using one of the get() methods.

Here is an example of how you can read data from a buffer into a channel:

//read from buffer into channel.

int bytesWritten = inChannel.write(buf);

Here is an example that reads data from a Buffer using the get() method:

byte aByte = buf.get();

**—-------------------------------------------------------------------------**

**package com.wipro.app;**

**import java.io.IOException;**

**import java.io.RandomAccessFile;**

**import java.nio.ByteBuffer;**

**import java.nio.channels.FileChannel;**

**public class NioChannelBufferDemo {**

**public static void main(String[] args) {**

**try {**

**RandomAccessFile aFile = new RandomAccessFile("/home/rps/Documents/data/javanotes.txt", "rw");**

**FileChannel inChannel = aFile.getChannel();**

**ByteBuffer buffer = ByteBuffer.*allocate*(48);**

**//channel writing data into buffer**

**// channel -> buffer**

**int bytesRead = inChannel.read(buffer);**

**System.*out*.println(bytesRead);**

**while (bytesRead != -1) {**

**// System.out.println("Read " + bytesRead);**

**/\***

**\* Channel has written 48 bytes of data into buffer using read() method,**

**\* now the buffer has to be flipped from write mode to read so that**

**\* we can read data from the buffer buffer.get() method.**

**\***

**\* flip() only to flip the buffer from write mode to read mode**

**\*/**

**buffer.flip();**

**while( buffer.hasRemaining() ){**

**System.*out*.print((char) buffer.get());**

**}**

**//buffer -> channel**

**// int bytesWritten = inChannel.write(buffer);**

**buffer.clear();**

**//channel writing data into buffer, channel -> buffer**

**bytesRead = inChannel.read(buffer);**

**}**

**aFile.close();**

**}catch(IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**—------------------------------------------------------**

**SELECTORS**

A selector provides a mechanism for monitoring one or more NIO channels and recognizing when one or more become available for data transfer.

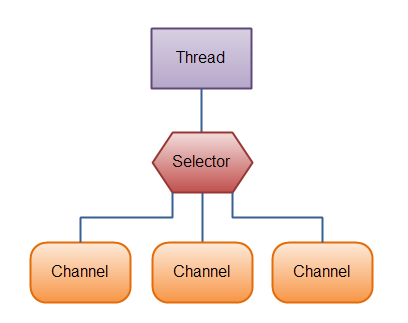
This way, **a single thread can be used for managing multiple channels**, and thus multiple network connections.

## **2. Why Use a Selector?**

With a selector, we can use one thread instead of several to manage multiple channels. **Context-switching between threads is expensive for the operating system**, and additionally, **each thread takes up memory.**

Therefore, the fewer threads we use, the better. However, it’s important to remember that **modern operating systems and CPU’s keep getting better at multitasking**, so the overheads of multi-threading keep diminishing over time.

**Note also that selectors not only help you read data; they can also listen for incoming network connections and write data across slow channels.**

****

1. **Creating a Selector**

**Selector selector = Selector.open();**

In order for a selector to monitor any channels, we must register these channels with the selector. We do this by invoking the *register* method of the selectable channel.

But before a channel is registered with a selector, it must be in non-blocking mode:

After that, we can register multiple channels with a selector object. When I/O activity happens on any of the channels, the selector notifies us. This is how we can read from a large number of data sources on a single thread.

1. **Registering Channels with the Selector**

*// channel has to be in non-blocking mode*

**channel.configureBlocking(false);**

**SelectionKey key = channel.register(selector, SelectionKey.OP\_READ);**

The first parameter is the *Selector* object we created earlier, the second parameter defines an interest set**,** meaning what events we are interested in listening for in the monitored channel, via the selector.

There are four different events we can listen for, each is represented by a constant in the *SelectionKey* class:

* *Connect* **–** when a client attempts to connect to the server. Represented by *SelectionKey.OP\_CONNECT*
* *Accept* **–** when the server accepts a connection from a client. Represented by *SelectionKey.OP\_ACCEPT*
* *Read* **–** when the server is ready to read from the channel. Represented by *SelectionKey.OP\_READ*
* *Write* **–** when the server is ready to write to the channel. Represented by *SelectionKey.OP\_WRITE*

The returned object *SelectionKey* represents the selectable channel’s registration with the selector.

**Interest set**

An interest set defines the set of events that we want the selector to watch out for on this channel. It is an integer value; we can get this information in the following way.

First, we have the interest set returned by the *SelectionKey*‘s *interestOps* method. Then we have the event constant in *SelectionKey* we looked at earlier.

When we AND these two values, we get a boolean value that tells us whether the event is being watched for or not:

**int interestSet = selectionKey.interestOps();**

**boolean isInterestedInAccept = interestSet & SelectionKey.OP\_ACCEPT;**

**boolean isInterestedInConnect = interestSet & SelectionKey.OP\_CONNECT;**

**boolean isInterestedInRead = interestSet & SelectionKey.OP\_READ;**

**boolean isInterestedInWrite = interestSet & SelectionKey.OP\_WRITE;**

### **The Ready Set**

The ready set defines the set of events that the channel is ready for. It is an integer value. We can use the following methods

**selectionKey.isAcceptable();**

**selectionKey.isConnectable();**

**selectionKey.isReadable();**

**selectionKey.isWriteable();**

### 

### **The Channel**

Accessing the channel being watched from the *SelectionKey* object is very simple. We just call the *channel* method:

**Channel channel = key.channel();**

### **The Selector**

Just like getting a channel, it’s very easy to obtain the *Selector* object from the *SelectionKey* object:

**Selector selector = key.selector();**

### **Attaching Objects**

We can attach an object to a *SelectionKey.* Sometimes we may want to give a channel a custom ID or attach any kind of Java object we may want to keep track of.

Here is how you attach and get objects from a *SelectionKey*:

**key.attach(Object);**

**Object object = key.attachment();**

Alternatively, we can choose to attach an object during channel registration. We add it as a third parameter to channel’s *register* method, like so:

**SelectionKey key = channel.register(**

**selector, SelectionKey.OP\_ACCEPT, object);**

## **Channel Key Selection**

So far, we have looked at how to create a selector, register channels to it and inspect the properties of the *SelectionKey* object which represents a channel’s registration to a selector.

This is only half of the process, now we have to perform a continuous process of selecting the ready set which we looked at earlier. We do selection using selector’s *select* method, like so:

**int channels = selector.select();**

This method blocks until at least one channel is ready for an operation. The integer returned represents the number of keys whose channels are ready for an operation.

Next, we usually retrieve the set of selected keys for processing:

**Set<SelectionKey> selectedKeys = selector.selectedKeys();**

The set we have obtained is of *SelectionKey* objects, each key represents a registered channel which is ready for an operation.

After this, we usually iterate over this set and for each key, we obtain the channel and perform any of the operations that appear in our interest set on it.

During the lifetime of a channel, it may be selected several times as its key appears in the ready set for different events.

This is why we must have a continuous loop to capture and process channel events as and when they occur.

we’re going to build a complete client-server example.

For ease of testing out our code, we’ll build an echo server and an echo client. In this kind of setup, the client connects to the server and starts sending messages to it. The server echoes back messages sent by each client.

When the server encounters a specific message, such as *end*, it interprets it as the end of the communication and closes the connection with the client.

### **The Server**

Here is our code for *EchoServer.java*:

public class EchoServer {

private static final String POISON\_PILL = "POISON\_PILL";

public static void main(String[] args) throws IOException {

Selector selector = Selector.open();

ServerSocketChannel serverSocket = ServerSocketChannel.open();

serverSocket.bind(new InetSocketAddress("localhost", 5454));

serverSocket.configureBlocking(false);

serverSocket.register(selector, SelectionKey.OP\_ACCEPT);

ByteBuffer buffer = ByteBuffer.allocate(256);

while (true) {

selector.select();

Set<SelectionKey> selectedKeys = selector.selectedKeys();

Iterator<SelectionKey> iter = selectedKeys.iterator();

while (iter.hasNext()) {

SelectionKey key = iter.next();

if (key.isAcceptable()) {

register(selector, serverSocket);

}

if (key.isReadable()) {

answerWithEcho(buffer, key);

}

iter.remove();

}

}

}

private static void answerWithEcho(ByteBuffer buffer, SelectionKey key)

throws IOException {

SocketChannel client = (SocketChannel) key.channel();

int r = client.read(buffer);

if (r == -1 || new String(buffer.array()).trim().equals(POISON\_PILL)) {

client.close();

System.out.println("Not accepting client messages anymore");

}

else {

buffer.flip();

client.write(buffer);

buffer.clear();

}

}

private static void register(Selector selector, ServerSocketChannel serverSocket)

throws IOException {

SocketChannel client = serverSocket.accept();

client.configureBlocking(false);

client.register(selector, SelectionKey.OP\_READ);

}

public static Process start() throws IOException, InterruptedException {

String javaHome = System.getProperty("java.home");

String javaBin = javaHome + File.separator + "bin" + File.separator + "java";

String classpath = System.getProperty("java.class.path");

String className = EchoServer.class.getCanonicalName();

ProcessBuilder builder = new ProcessBuilder(javaBin, "-cp", classpath, className);

return builder.start();

}

}

This is what is happening; we create a *Selector* object by calling the static *open* method. We then create a channel also by calling its static *open* method, specifically a *ServerSocketChannel* instance.

This is because ***ServerSocketChannel* is selectable and good for a stream-oriented listening socket**.

We then bind it to a port of our choice. Remember we said earlier that before registering a selectable channel to a selector, we must first set it to non-blocking mode. So next we do this and then register the channel to the selector.

We don’t need the *SelectionKey* instance of this channel at this stage, so we will not remember it.

Java NIO uses a buffer-oriented model other than a stream-oriented model. So socket communication usually takes place by writing to and reading from a buffer.

We, therefore, create a new *ByteBuffer* which the server will be writing to and reading from. We initialize it to 256 bytes, it’s just an arbitrary value, depending on how much data we plan to transfer to and fro.

Finally, we perform the selection process. We select the ready channels, retrieve their selection keys, iterate over the keys, and perform the operations for which each channel is ready.

We do this in an infinite loop since servers usually need to keep running whether there is an activity or not.

The only operation a *ServerSocketChannel* can handle is an *ACCEPT* operation. When we accept the connection from a client, we obtain a *SocketChannel* object on which we can do read and writes. We set it to non-blocking mode and register it for a READ operation to the selector.

During one of the subsequent selections, this new channel will become read-ready. We retrieve it and read it contents into the buffer. True to it’s as an echo server, we must write this content back to the client.

**When we desire to write to a buffer from which we have been reading, we must call the *flip()* method**.

We finally set the buffer to write mode by calling the *flip* method and simply write to it.

The *start()* method is defined so that the echo server can be started as a separate process during unit testing.

### **The Client**

Here is our code for *EchoClient.java*:

public class EchoClient {

private static SocketChannel client;

private static ByteBuffer buffer;

private static EchoClient instance;

public static EchoClient start() {

if (instance == null)

instance = new EchoClient();

return instance;

}

public static void stop() throws IOException {

client.close();

buffer = null;

}

private EchoClient() {

try {

client = SocketChannel.open(new InetSocketAddress("localhost", 5454));

buffer = ByteBuffer.allocate(256);

} catch (IOException e) {

e.printStackTrace();

}

}

public String sendMessage(String msg) {

buffer = ByteBuffer.wrap(msg.getBytes());

String response = null;

try {

client.write(buffer);

buffer.clear();

client.read(buffer);

response = new String(buffer.array()).trim();

System.out.println("response=" + response);

buffer.clear();

} catch (IOException e) {

e.printStackTrace();

}

return response;

}

}

The client is simpler than the server.

We use a singleton pattern to instantiate it inside the *start* static method. We call the private constructor from this method.

In the private constructor, we open a connection on the same port on which the server channel was bound and still on the same host.

We then create a buffer to which we can write and from which we can read.

Finally, we have a *sendMessage* method which reads wraps any string we pass to it into a byte buffer which is transmitted over the channel to the server.

We then read from the client channel to get the message sent by the server. We return this as the echo of our message.

**MAVEN Build Tool**

**Maven is one of the popular build tools. ANT, Gradle are two other popular build tools.**

Building a software project typically consists of such tasks as downloading dependencies, putting additional jars on a classpath, compiling source code into binary code, running tests, packaging compiled code into deployable artifacts such as JAR, WAR, and ZIP files, and deploying these artifacts to an application server or repository.

[Apache Maven](https://maven.apache.org/) automates these tasks, minimizing the risk of humans making errors while building the software manually and separating the work of compiling and packaging our code from that of code construction.

## **Why Use Maven?**

The key features of Maven are:

* **simple project setup that follows best practices:** Maven tries to avoid as much configuration as possible, by supplying project templates (named *archetypes*)
* **dependency management:** it includes automatic updating, downloading and validating the compatibility, as well as reporting the dependency closures (known also as transitive dependencies)
* **isolation between project dependencies and plugins:** with Maven, project dependencies are retrieved from the *dependency repositories* while any plugin’s dependencies are retrieved from the *plugin repositories,* resulting in fewer conflicts when plugins start to download additional dependencies
* **central repository system:** project dependencies can be loaded from the local file system or public repositories, such as [**Maven Central**](https://search.maven.org/)

## **Project Object Model**

The configuration of a Maven project is done via a *Project Object Model (POM)*, represented by a *pom.xml* file. The *POM* describes the project, manages dependencies, and configures plugins for building the software.

<project>

<modelVersion>4.0.0</modelVersion>

<groupId>com.baeldung</groupId>

<artifactId>baeldung</artifactId>

<packaging>jar</packaging>

<version>1.0-SNAPSHOT</version>

<name>com.baeldung</name>

<url>http://maven.apache.org</url>

<dependencies>

<dependency>

<groupId>org.junit.jupiter</groupId>

<artifactId>junit-jupiter-api</artifactId>

<version>5.8.2</version>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

//...

</plugin>

</plugins>

</build>

</project>

### **Project Identifiers**

Maven uses a set of identifiers, also called coordinates, to uniquely identify a project and specify how the project artifact should be packaged:

* *groupId* – a unique base name of the company or group that created the project
* *artifactId* – a unique name of the project
* *version* – a version of the project
* *packaging* – a packaging method (e.g. *WAR*/*JAR*/*ZIP*)

The first three of these (*groupId:artifactId:version*) combine to form the unique identifier and are the mechanism by which you specify which versions of external libraries (e.g. JARs) your project will use.

### **Dependencies**

These are external libraries that a project uses are called dependencies. The dependency management feature in Maven ensures the automatic download of those libraries from a central repository,

Ex.

<**dependency**>

<**groupId**>org.springframework</**groupId**>

<**artifactId**>spring-core</**artifactId**>

<**version**>5.3.16</**version**>

</**dependency**>

A repository in Maven is used to hold build artifacts and dependencies of varying types. The default local repository is located in the .m2/repository folder under the home directory of the user.

If an artifact or a plugin is available in the local repository, Maven uses it. Otherwise, it is downloaded from a central repository and stored in the local repository. The default central repository is

[**https://mvnrepository.com/**](https://mvnrepository.com/)

**Note: Dependencies are jar files which are required to execute/run the project. They are part of final build.**

### **Properties**

Custom properties can help to make your *pom.xml* file easier to read and maintain.

**<properties>**

**<spring.version>5.3.16</spring.version>**

**</properties>**

**<dependencies>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-core</artifactId>**

**<version>${spring.version}</version>**

**</dependency>**

**<dependency>**

**<groupId>org.springframework</groupId>**

**<artifactId>spring-context</artifactId>**

**<version>${spring.version}</version>**

**</dependency>**

**</dependencies>**

### **Build**

The *build* section is also a very important section of the Maven *POM.* It provides information about the default Maven *goal*, the directory for the compiled project, and the final name of the application. The default *build* section looks like this:

**<build>**

**<defaultGoal>install</defaultGoal>**

**<directory>${basedir}/target</directory>**

**<finalName>${artifactId}-${version}</finalName>**

**<filters>**

**<filter>filters/filter1.properties</filter>**

**</filters>**

**//...**

**</build>**

### **Using *Profiles***

Another important feature of Maven is its support for *profiles.* **A *profile* is basically a set of configuration values.** By using *profiles*, you can customize the build for different environments such as **Production/Test/Development.**

<profiles>

<profile>

<id>production</id>

<build>

<plugins>

<plugin>

//...

</plugin>

</plugins>

</build>

</profile>

<profile>

<id>development</id>

<activation>

<activeByDefault>true</activeByDefault>

</activation>

<build>

<plugins>

<plugin>

//...

</plugin>

</plugins>

</build>

</profile>

</profiles>

## **Maven Build Lifecycle**

Every Maven build follows a specified *lifecycle*. You can execute several build *lifecycle* *goals*, including the ones to *compile* the project’s code, create a *package,* and *install* the archive file in the local Maven dependency repository.

### ***Lifecycle Phases***

The following list shows the most important Maven *lifecycle* phases:

* *validate* – checks the correctness of the project
* *compile* – compiles the provided source code into binary artifacts
* *test* – executes unit tests
* *package* – packages compiled code into an archive file
* *integration-test* – executes additional tests, which require the packaging
* *verify* – checks if the package is valid
* *install* – installs the package file into the local Maven repository
* *deploy* – deploys the package file to a remote server or repository

When we execute the following command

**mvn test**

The previous lifecycle phases will get executed:

**validate, compile and test**

If we execute **maven install, validate, compile, test, package, integration-test, verify and install phases** will be executed.

#### **Maven Clean Lifecycle:**

1. The Maven clean lifecycle consists of the following phases:

* pre-clean
* clean
* Post-clean

2. By using the Clean phase (***mvn clean*), you ensure a clean and consistent build environment. It removes previously generated artifacts, reducing the risk of conflicts or outdated files.**

**Note: To clean and build the maven project, execute the following command:**

**mvn clean install or mvn clean package**

### ***Plugins* and *Goals***

A Maven *plugin* is a collection of one or more *goals*. Goals are executed in phases, which helps to determine the order in which the *goals* are executed.

### **Understanding Maven Goals and Plugins**

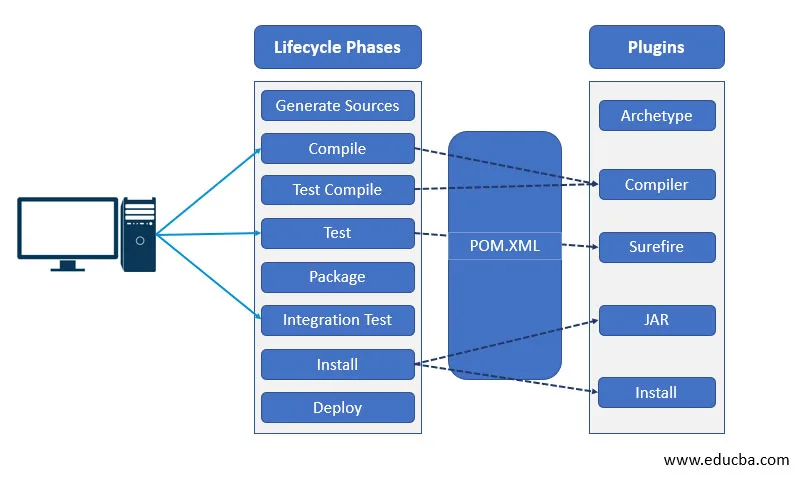
In Maven, goals and plugins play a crucial role in executing specific tasks and extending the functionality of the build process.

#### Maven Goals:

* Goals are specific tasks that Maven can perform during the build lifecycle.
* Each phase of the lifecycle is associated with one or more goals.
* Maven provides default goals for each phase, but you can also define custom goals.
* Goals can be executed from the command line using the *mvn* command followed by the goal name.
* Examples of common Maven goals include compile, test, package, and install.

#### Maven Plugins:

* Plugins are the building blocks of Maven’s functionality.
* They provide implementations for various goals and can be used to extend the build process.
* Maven has a vast ecosystem of plugins that cover a wide range of tasks and integrations.
* Plugins can be configured in the project’s POM file, specifying the desired version and any custom configurations.
* Maven resolves and downloads plugins from remote repositories when needed.
* Examples of popular Maven plugins include the Maven Compiler Plugin, Surefire Plugin for testing, and the Maven Assembly Plugin for creating custom distributions.



**Maven Installation:**

**sudo apt install maven**

**mvn -version**

**Maven Cheat Sheet**



<https://maven.apache.org/guides/getting-started/maven-in-five-minutes.html>

**Creating Java project using Maven build at CLI**

## **Creating a Project**

You need somewhere for your project to reside. Create a directory somewhere and start a shell in that directory. On your command line, execute the following Maven goal:

mvn archetype:generate -DgroupId=com.mycompany.app -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false

**Ex.**

**mvn archetype:generate -DgroupId=com.wipro -DartifactId=my-app -DarchetypeArtifactId=maven-archetype-quickstart -DarchetypeVersion=1.4 -DinteractiveMode=false**

### **Build the Project**

**mvn package**

**Test the Project**

**You may test the newly compiled and packaged JAR with the following command:**

**java -cp target/my-app-1.0-SNAPSHOT.jar com.wipro.App**

**Which will print the quintessential:**

**Hello World!**

**Note:**

## **Java 9 or later**

**By default your version of Maven might use an old version of the maven-compiler-plugin that is not compatible with Java 9 or later versions. To target Java 9 or later, you should at least use version 3.6.0 of the maven-compiler-plugin and set the maven.compiler.release property to the Java release you are targetting (e.g. 9, 10, 11, 12, etc.).**

**In the following example, we have configured our Maven project to use version 3.8.1 of maven-compiler-plugin and target Java 11:**

1. **<properties>**
2. **<maven.compiler.release>11</maven.compiler.release>**
3. **</properties>**
5. **<build>**
6. **<pluginManagement>**
7. **<plugins>**
8. **<plugin>**
9. **<groupId>org.apache.maven.plugins</groupId>**
10. **<artifactId>maven-compiler-plugin</artifactId>**
11. **<version>3.8.1</version>**
12. **</plugin>**
13. **</plugins>**
14. **</pluginManagement>**
15. **</build>**

**To run the jar file using java -jar command, we need to specify the maven-jar-plugin the packageName.className that contains main() method.**

**<build>**

**<plugins>**

**<plugin>**

**<artifactId>maven-jar-plugin</artifactId>**

**<version>3.0.2</version>**

**<configuration>**

**<archive>**

**<manifest>**

**<mainClass>com.wipro.App</mainClass>**

**</manifest>**

**</archive>**

**</configuration>**

**</plugin>**

**</plugins>**

**</build>**

**Rebuild the project.**

Run the following command from project’s root folder

**mvn clean package**

Change directory to target folder (cd target) and execute the following command:

**rps@rps-virtual-machine:~/Documents/java\_apps/my-app/target$ java -jar my-app-1.0-SNAPSHOT.jar**

**Hello World!**

**—------------------------------------------------------------------------------------**

**To change the default jar filename,add the following in the build section of pom.xml file**

<build>

**<finalName>my-app</finalName>**

**—------------------------------**

</build>

—-----------------------------------------------------

**package** com.wipro.model;

**public** **class** HelloWorld {

**private** String greeting;

**public** HelloWorld() {

}

**public** HelloWorld(String greeting) {

**super**();

**this**.greeting = greeting;

}

**public** String getGreeting() {

**return** greeting;

}

**public** **void** setGreeting(String greeting) {

**this**.greeting = greeting;

}

@Override

**public** String toString() {

**return** "HelloWorld [greeting=" + greeting + "]";

}

}

**package com.wipro;**

**import java.time.LocalDateTime;**

**import com.wipro.model.HelloWorld;**

**public class App {**

**public static void main( String[] args ){**

**HelloWorld helloWorld = new HelloWorld("Hi, Now time is "+ LocalDateTime.*now*());**

**System.*out*.println(helloWorld.getGreeting());**

**}**

**}**

**<dependency>**

**<groupId>org.junit.jupiter</groupId>**

**<artifactId>junit-jupiter-engine</artifactId>**

**<version>5.9.1</version>**

**<scope>test</scope>**

**</dependency>**

**package com.wipro.model;**

**import static org.junit.jupiter.api.Assertions.\*;**

**import org.junit.jupiter.api.AfterEach;**

**import org.junit.jupiter.api.BeforeEach;**

**import org.junit.jupiter.api.Test;**

**class HelloWorldTest {**

**private HelloWorld helloWorld=null;**

**@BeforeEach**

**void setUp() throws Exception {**

**helloWorld = new HelloWorld("Welcome to JUnit5");**

**}**

**@AfterEach**

**void tearDown() throws Exception {**

**helloWorld=null;**

**}**

**@Test**

**void testHelloWorldString() {**

**HelloWorld testHelloWorld = new HelloWorld("Welcome");**

**assertEquals("Welcome",testHelloWorld.getGreeting());**

**}**

**@Test**

**void testGetGreetingForTrue() {**

**assertEquals("Welcome to JUnit5",helloWorld.getGreeting());**

**}**

**@Test**

**void testGetGreetingForFalse() {**

**assertEquals("Welcome",helloWorld.getGreeting());**

**}**

**@Test**

**void testSetGreetingForTrue() {**

**helloWorld.setGreeting("Good Morning");**

**assertEquals("Good Morning",helloWorld.getGreeting());**

**}**

**@Test**

**void testSetGreetingForFalse() {**

**helloWorld.setGreeting("Good Morning");**

**assertNotEquals("Good Day",helloWorld.getGreeting());**

**}**

**}**

**<profiles>**

**<profile>**

**<id>no-tests</id>**

**<properties>**

**<maven.test.skip>true</maven.test.skip>**

**</properties>**

**</profile>**

**<profile>**

**<id>dev</id>**

**<activation>**

**<activeByDefault>true</activeByDefault>**

**</activation>**

**<build>**

**<finalName>my-app-dev</finalName>**

**</build>**

**</profile>**

**<profile>**

**<id>prod</id>**

**<build>**

**<finalName>my-app-prod</finalName>**

**</build>**

**</profile>**

**<profile>**

**<id>test</id>**

**<build>**

**<finalName>my-app-test</finalName>**

**</build>**

**</profile>**

**</profiles>**

**—--------------------------------------------------------------------------------**

**<properties>**

**<project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>**

**<maven.compiler.source>22</maven.compiler.source>**

**<maven.compiler.target>22</maven.compiler.target>**

**</properties>**

**<dependencies>**

**<dependency>**

**<groupId>org.junit.jupiter</groupId>**

**<artifactId>junit-jupiter-engine</artifactId>**

**<version>5.9.1</version>**

**<scope>test</scope>**

**</dependency>**

**</dependencies>**

**—--------------------------------------------------------------------------**

**Oracle JDBC Driver Maven Dependency**

**<dependency>**

**<groupId>com.oracle.database.jdbc</groupId>**

**<artifactId>ojdbc6</artifactId>**

**<version>11.2.0.4</version>**

**</dependency>**

**Note: We need to start oracle listener before running the application,**

**Go to the terminal and run the following command:**

**c**

**package com.wipro.util;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**import java.util.TimeZone;**

**public class OracleConnection {**

**static{**

**try {**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

**} catch (ClassNotFoundException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**}**

**}**

**public static Connection getConnection(){**

**try {**

**TimeZone timeZone = TimeZone.*getTimeZone*("Asia/Kolkata");**

**TimeZone.*setDefault*(timeZone);**

**Connection connection =**

**DriverManager.*getConnection*("jdbc:oracle:thin:@localhost:9501/XE","system","rps@123");**

**return connection;**

**} catch (SQLException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**return null;**

**}**

**}**

**}**

**package com.wipro.app;**

**import java.sql.Connection;**

**import com.wipro.util.OracleConnection;**

**public class App {**

**public static void main( String[] args ){**

**try {**

**Connection connection = OracleConnection.*getConnection*();**

**if(connection == null) {**

**throw new Exception("Unable to connect to the database");**

**}**

**System.*out*.println("Connected to Oracle database");**

**}catch(Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**Note:**

**Before running the application, in project's Build Path, set the VM arguments as:**

**-Doracle.jdbc.timezoneAsRegion=false**

**Run As -> Run Configuration, update the configuration, by placing the above argument in VM Arguments.**

**—-----------------------------------------------------------------------------------------------------------**

**package com.wipro.app;**

**import java.sql.Connection;**

**import java.sql.ResultSet;**

**import java.sql.Statement;**

**import com.wipro.util.OracleConnection;**

**public class App {**

**public static void main( String[] args ){**

**// try {**

**// Connection connection = OracleConnection.getConnection();**

**// if(connection == null) {**

**// throw new Exception("Unable to connect to the database");**

**// }**

**//**

**// System.out.println("Connected to Oracle database");**

**// }catch(Exception e) {**

**// e.printStackTrace();**

**// }**

**String sql = "select \* from emp";**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**Statement statement = connection.createStatement();**

**){**

**ResultSet resultSet = statement.executeQuery(sql);**

**while(resultSet.next()) {**

**System.*out*.println(resultSet.getInt("ID")+","+resultSet.getString("NAME")**

**+ ","+resultSet.getInt("AGE")+ ","+resultSet.getDouble("SALARY")+ ","+ resultSet.getInt("DEPTID"));**

**}**

**}catch(Exception e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**—-----------------------------------------------------------------------------------------------------------**

**create table person(**

**adhar\_card number(12) primary key,**

**name varchar2(30) not null,**

**gender varchar2(10),**

**birthdate date,**

**address varchar2(100),**

**mobile number(10)**

**)**

**insert into person values(666161233315,'Priya','FEMALE','15-JUL-2000','Hyderabad',7967078999);**

**select \* from person;**

**—-------------------------------database.properties—--------------------------------------**

**ORACLE\_URL = jdbc:oracle:thin:@localhost:9501/XE**

**ORACLE\_USER = system**

**ORACLE\_PASSWORD = rps@123**

**—------------------------------OracleConnection.java—--------------------------------------**

**package com.wipro.util;**

**import java.io.FileInputStream;**

**import java.io.IOException;**

**import java.sql.Connection;**

**import java.sql.DriverManager;**

**import java.sql.SQLException;**

**import java.util.Properties;**

**import java.util.TimeZone;**

**public class OracleConnection {**

**private static Properties *properties*=null;**

**static{**

**try(**

**FileInputStream fis= new FileInputStream("");**

**){**

**Class.*forName*("oracle.jdbc.driver.OracleDriver");**

***properties*= new Properties();**

***properties*.load(fis);**

**} catch (ClassNotFoundException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**} catch(IOException e) {**

**e.printStackTrace();**

**}**

**}**

**public static Connection getConnection(){**

**try {**

**TimeZone timeZone = TimeZone.*getTimeZone*("Asia/Kolkata");**

**TimeZone.*setDefault*(timeZone);**

**Connection connection =**

**DriverManager.*getConnection*(*properties*.getProperty("ORACLE\_URL"),**

***properties*.getProperty("ORACLE\_USER"),**

***properties*.getProperty("ORACLE\_PASSWORD"));**

**return connection;**

**} catch (SQLException e) {**

**// TODO Auto-generated catch block**

**e.printStackTrace();**

**return null;**

**}**

**}**

**}**

**—------------------------------------------------GenderTyp.java—---------------------------**

**package com.wipro.model;**

**public enum GenderTyp {**

***MALE*, *FEMALE*, *TRANSGENDER***

**}**

**—-------------------------------------------------Person.java—----------------------------------**

**package com.wipro.model;**

**import java.time.LocalDate;**

**/\***

**\* A POJO class which is called as domain/DTO/VO object**

**\*/**

**public class Person {**

**private Long adharCard;**

**private String name;**

**private GenderTyp gender;**

**private LocalDate birthdate;**

**private String address;**

**private Long mobile;**

**public Person() {**

**}**

**public Person(Long adharCard, String name, GenderTyp gender, LocalDate birthdate, String address, Long mobile) {**

**super();**

**this.adharCard = adharCard;**

**this.name = name;**

**this.gender = gender;**

**this.birthdate = birthdate;**

**this.address = address;**

**this.mobile = mobile;**

**}**

**public Long getAdharCard() {**

**return adharCard;**

**}**

**public void setAdharCard(Long adharCard) {**

**this.adharCard = adharCard;**

**}**

**public String getName() {**

**return name;**

**}**

**public void setName(String name) {**

**this.name = name;**

**}**

**public GenderTyp getGender() {**

**return gender;**

**}**

**public void setGender(GenderTyp gender) {**

**this.gender = gender;**

**}**

**public LocalDate getBirthdate() {**

**return birthdate;**

**}**

**public void setBirthdate(LocalDate birthdate) {**

**this.birthdate = birthdate;**

**}**

**public String getAddress() {**

**return address;**

**}**

**public void setAddress(String address) {**

**this.address = address;**

**}**

**public Long getMobile() {**

**return mobile;**

**}**

**public void setMobile(Long mobile) {**

**this.mobile = mobile;**

**}**

**@Override**

**public String toString() {**

**return "Person [adharCard=" + adharCard + ", name=" + name + ", gender=" + gender + ", birthdate=" + birthdate**

**+ ", address=" + address + ", mobile=" + mobile + "]";**

**}**

**}**

**—------------------PersonDAO—--------------------------------------------------------------**

**package com.wipro.dao;**

**import java.sql.SQLException;**

**import java.util.List;**

**import com.wipro.model.Person;**

**//CRUD : Create, Read, Update , Delete operations**

**public interface PersonDAO {**

**public abstract String addPerson(Person person) throws SQLException;**

**public abstract String updatePerson(Person person) throws SQLException;**

**public abstract String deletePerson(Long adharCard ) throws SQLException;**

**public abstract Person getPersonById(Long adharCard ) throws SQLException;**

**public abstract List<Person> getAllPersons() throws SQLException;**

**}**

**—-----------------------PersonDaoImpl.java—-------------------------------**

**package com.wipro.dao;**

**import java.sql.SQLException;**

**import java.util.List;**

**import com.wipro.model.Person;**

**public class PersonDaoImpl implements PersonDAO{**

**@Override**

**public String addPerson(Person person) throws SQLException {**

**String sql = "insert into person values(?,?,?,?,?,?)";**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**PreparedStatement preparedStatement = connection.prepareCall(sql);**

**){**

**preparedStatement.setLong(1, person.getAdharCard());**

**preparedStatement.setString(2, person.getName());**

**preparedStatement.setString(3, person.getGender().toString());**

**//convert java.time.LocalDate to java.sql.Date**

**preparedStatement.setDate(4, java.sql.Date.*valueOf*(person.getBirthdate()));**

**preparedStatement.setString(5, person.getAddress());**

**preparedStatement.setLong(6,person.getMobile());**

**int n = preparedStatement.executeUpdate();**

**if(n==0) {**

**throw new SQLException("Unable to add person");**

**}**

**return "ADDED PERSON TO DATABASE";**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**@Override**

**public String updatePerson(Person person) throws SQLException {**

**String sql = "update person set name=?,gender=?,birthdate=?,address=?,mobile=? where adhar\_card=?";**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**PreparedStatement preparedStatement = connection.prepareCall(sql);**

**){**

**preparedStatement.setString(1, person.getName());**

**preparedStatement.setString(2, person.getGender().toString());**

**//convert java.time.LocalDate to java.sql.Date**

**preparedStatement.setDate(3, java.sql.Date.*valueOf*(person.getBirthdate()));**

**preparedStatement.setString(4, person.getAddress());**

**preparedStatement.setLong(5, person.getMobile());**

**preparedStatement.setLong(6, person.getAdharCard());**

**int n = preparedStatement.executeUpdate();**

**if(n==0) {**

**throw new SQLException("Unable be up update the person data");**

**}**

**return "PERSON:"+adharCard+ " UPDATED TO DATABASE";**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**@Override**

**public String deletePerson(Long adharCard) throws SQLException {**

**String sql = "delete from person where adhar\_card=?";**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**PreparedStatement preparedStatement = connection.prepareCall(sql);**

**){**

**preparedStatement.setLong(1, adharCard);**

**int n = preparedStatement.executeUpdate();**

**if(n==0) {**

**throw new SQLException("Invalid AdharCard");**

**}**

**return "PERSON:"+person.getAdharCard()+ " DELETED FROM DATABASE";**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**@Override**

**public Person getPersonById(Long adharCard) throws SQLException {**

**String sql = "select \* from person where adhar\_card=?";**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**PreparedStatement preparedStatement = connection.prepareCall(sql);**

**){**

**preparedStatement.setLong(1, adharCard);**

**ResultSet resultSet = preparedStatement.executeQuery();**

**if(resultSet.next()) {**

**Person person = new Person();**

**populate(person,resultSet);**

**return person;**

**}else {**

**throw new SQLException("Invalid AdharCard Number")**

**}**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**@Override**

**public List<Person> getAllPersons() throws SQLException {**

**String sql = "select \* from person";**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**Statement statement = connection.createStatement();**

**){**

**ResultSet resultSet = statement.executeQuery(sql);**

**List<Person> personList = new ArrayList<>();**

**while(resultSet.next()) {**

**Person person = new Person();**

**//copy contents of the current row of the resultSet object into person object**

**populate(person,resultSet);**

**//add person object to the list**

**personList.add(person);**

**}**

**return personList;**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**}**

**private void populate(Person person, ResultSet resultSet) throws SQLException {**

**person.setAdharCard(resultSet.getLong("ADHAR\_CARD"));**

**person.setName(resultSet.getString("NAME"));**

**//convert String -> GenderTyp**

**person.setGender( GenderTyp.*valueOf*(resultSet.getString("GENDER")));**

**//convert java.sql.Date to java.time.LocalDate**

**person.setBirthdate(resultSet.getDate("BIRTHDATE").toLocalDate());**

**person.setAddress(resultSet.getString("ADDRESS"));**

**person.setMobile(resultSet.getLong("MOBILE"));**

**}**

**—---------------------------------------------------------------------------------------------**

**To execute any SQL statement(insert/update/delete/select), JDBC API**

**provides 3 interfaces.**

**CallableStatement -> PreparedStatement -> Statement**

**Statement:**

**Use Statement interface to execute DDL statements (create/alter/drop) and static queries.**

**A query that returns same result every time it is executed is called as static query. A query without where clause is a static query.**

**Ex.**

**select \* from person;**

**PreparedStatement:**

**Use PreparedStatement interface for DML statements(insert/update/delete) and dynamic queries.**

**A query that has where clause is a dynamic query.**

**A query that returns different results whenever it is executed is called as dynamic query.**

**CallableStatement**

**Use CallableStatement to call sub programs(procedures/functions) of the database.**

**Exception handling in layered architecture**

Exceptions have to be explicitly handled in the presentation layer.

The DAO and Service layers have to throw exceptions to the presentation layer.

The methods of DAO layer throw SQL exceptions which have to be captured in Service layer, have to be customized and then throw custom exceptions to the presentation layer.

Service layer converts SQLException to Custom Exception and throws custom exceptions to presentation layer.

—------------------com.wipro.exception.PersonException—-----------------------

**package com.wipro.exception;**

**//custom checked exception**

**public class PersonException extends Exception{**

**public PersonException() {**

**super("Person Exception Thrown");**

**}**

**public PersonException(String message) {**

**super(message);**

**}**

**public PersonException(String message, Throwable t) {**

**super(message,t);**

**}**

**}**

**—-----------------------------------------com.wipro.service.PersonService—-------------**

**package com.wipro.service;**

**import java.util.List;**

**import com.wipro.exception.PersonException;**

**import com.wipro.model.Person;**

**public interface PersonService {**

**public abstract String addPerson(Person person) throws PersonException;**

**public abstract String updatePerson(Person person) throws PersonException;**

**public abstract String deletePerson(Long adharCard ) throws PersonException;**

**public abstract Person getPersonById(Long adharCard ) throws PersonException;**

**public abstract List<Person> getAllPersons() throws PersonException;**

**}**

**—------------------------------------------------------------------------------**

**package com.wipro.service;**

**import java.sql.SQLException;**

**import java.util.List;**

**import com.wipro.dao.PersonDAO;**

**import com.wipro.dao.PersonDaoImpl;**

**import com.wipro.exception.PersonException;**

**import com.wipro.model.Person;**

**public class PersonServiceImpl implements PersonService{**

**private PersonDAO personDAO = new PersonDaoImpl();**

**@Override**

**public String addPerson(Person person) throws PersonException {**

**try {**

**return personDAO.addPerson(person);**

**}catch(SQLException e) {**

**throw new PersonException(e.getMessage(),e);**

**}**

**}**

**@Override**

**public String updatePerson(Person person) throws PersonException {**

**try {**

**String message = personDAO.updatePerson(person);**

**return message;**

**}catch(SQLException e) {**

**throw new PersonException(e.getMessage(),e);**

**}**

**}**

**@Override**

**public String deletePerson(Long adharCard) throws PersonException {**

**try {**

**String message = personDAO.deletePerson(adharCard);**

**return message;**

**}catch(SQLException e) {**

**throw new PersonException(e.getMessage(),e);**

**}**

**}**

**@Override**

**public Person getPersonById(Long adharCard) throws PersonException {**

**try {**

**Person person = personDAO.getPersonById(adharCard);**

**return person;**

**}catch(SQLException e) {**

**throw new PersonException(e.getMessage(),e);**

**}**

**}**

**@Override**

**public List<Person> getAllPersons() throws PersonException {**

**try {**

**List<Person> personList = personDAO.getAllPersons();**

**//post-processing working**

**return personList;**

**}catch(SQLException e) {**

**throw new PersonException(e.getMessage(),e);**

**}**

**}**

**}**

**—-------------------------------------------------------------------------------**

**package com.wipro.app;**

**import java.time.LocalDate;**

**import java.util.List;**

**import java.util.Scanner;**

**import com.wipro.exception.PersonException;**

**import com.wipro.model.GenderTyp;**

**import com.wipro.model.Person;**

**import com.wipro.service.PersonService;**

**import com.wipro.service.PersonServiceImpl;**

**public class App {**

**private static PersonService *personService* = new PersonServiceImpl();**

**private static Scanner *scanner* = new Scanner(System.*in*);**

**public static void main( String[] args ){**

**// try {**

**// Person person = new Person(987567487612L,"Anjali",**

**// GenderTyp.FEMALE,LocalDate.of(2001, 10, 21),"Pune",7693265789L);**

**//**

**// String message = personService.addPerson(person);**

**// System.out.println(message);**

**//**

**// }catch(PersonException e) {**

**// e.printStackTrace();**

**// }**

**try {**

**List<Person> personList = *personService*.getAllPersons();**

**personList.forEach(System.*out*::println);**

**}catch(PersonException e) {**

**e.printStackTrace();**

**}**

**// try {**

**// System.out.println("Enter adharcard: ");**

**// long adharCard = Long.parseLong(scanner.nextLine());**

**// Person person = personService.getPersonById(adharCard);**

**// System.out.println(person);**

**//**

**// }catch(PersonException e) {**

**// e.printStackTrace();**

**// }**

**// try {**

**// System.out.println("Enter adharcard of the person to delete: ");**

**// long adharCard = Long.parseLong(scanner.nextLine());**

**// String message = personService.deletePerson(adharCard);**

**// System.out.println(message);**

**//**

**// }catch(PersonException e) {**

**// e.printStackTrace();**

**// }**

**try {**

**System.*out*.println("Enter adharcard of the person to update: ");**

**long adharCard = Long.*parseLong*(*scanner*.nextLine());**

**Person person = *personService*.getPersonById(adharCard);**

**System.*out*.println("Hi, "+ person.getName()+ "Do want to change your name(yes/no)");**

**String option = *scanner*.nextLine();**

**if(option.equalsIgnoreCase("yes")) {**

**System.*out*.println("Enter your new name: ");**

**person.setName(*scanner*.nextLine());**

**}**

**System.*out*.println("Change Gender? (yes/no): ");**

**option = *scanner*.nextLine();**

**if(option.equalsIgnoreCase("yes")) {**

**System.*out*.println("Enter Gender ");**

**person.setGender( GenderTyp.*valueOf*(*scanner*.nextLine()));**

**}**

**System.*out*.println("Change Birthdate? (yes/no): ");**

**option = *scanner*.nextLine();**

**if(option.equalsIgnoreCase("yes")) {**

**System.*out*.println("Enter New Birthdate(yyyy-mm-dd) ");**

**//String -> LocalDate**

**person.setBirthdate(LocalDate.*parse*(*scanner*.nextLine()));**

**}**

**System.*out*.println("Change Address? (yes/no): ");**

**option = *scanner*.nextLine();**

**if(option.equalsIgnoreCase("yes")) {**

**System.*out*.println("Enter New Address: ");**

**//String -> LocalDate**

**person.setAddress(*scanner*.nextLine());**

**}**

**System.*out*.println("Change Mobile? (yes/no): ");**

**option = *scanner*.nextLine();**

**if(option.equalsIgnoreCase("yes")) {**

**System.*out*.println("Enter New Mobile N umber: ");**

**person.setMobile(Long.*parseLong*(*scanner*.nextLine()));**

**}**

**String message = *personService*.updatePerson(person);**

**System.*out*.println(message);**

**}catch(PersonException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

**—---------------------------------------------------------------------------------------------------------------------------**

**package com.wipro.dao;**

**/\***

**\* An interface can contain public static final varaibles**

**\*/**

**public interface QueryMapper {**

**public static final String *ADD\_PERSON* = "insert into person values(?,?,?,?,?,?)";**

**public static final String *GET\_PERSON\_BY\_ID* = "select \* from person where adhar\_card=?";**

**public static final String *GET\_ALL\_PERSONS* = "select \* from person";**

**public static final String *DELETE\_PERSON* = "delete from person where adhar\_card=?";**

**public static final String *UPDATE\_PERSON* = "update person set name=?,gender=?,birthdate=?,address=?,mobile=? where adhar\_card=?";**

**}**

**—--------------------------------------------------------------------------------------------------**

**package com.wipro.dao;**

**import java.sql.Connection;**

**import java.sql.PreparedStatement;**

**import java.sql.ResultSet;**

**import java.sql.SQLException;**

**import java.sql.Statement;**

**import java.util.ArrayList;**

**import java.util.List;**

**import com.wipro.model.GenderTyp;**

**import com.wipro.model.Person;**

**import com.wipro.util.OracleConnection;**

**public class PersonDaoImpl implements PersonDAO{**

**@Override**

**public String addPerson(Person person) throws SQLException {**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**PreparedStatement preparedStatement = connection.prepareCall(QueryMapper.*ADD\_PERSON*);**

**){**

**preparedStatement.setLong(1, person.getAdharCard());**

**preparedStatement.setString(2, person.getName());**

**preparedStatement.setString(3, person.getGender().toString());**

**//convert java.time.LocalDate to java.sql.Date**

**preparedStatement.setDate(4, java.sql.Date.*valueOf*(person.getBirthdate()));**

**preparedStatement.setString(5, person.getAddress());**

**preparedStatement.setLong(6,person.getMobile());**

**int n = preparedStatement.executeUpdate();**

**if(n==0) {**

**throw new SQLException("Unable to add person");**

**}**

**return "ADDED PERSON TO DATABASE";**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**@Override**

**public String updatePerson(Person person) throws SQLException {**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**PreparedStatement preparedStatement = connection.prepareCall(QueryMapper.*UPDATE\_PERSON*);**

**){**

**preparedStatement.setString(1, person.getName());**

**preparedStatement.setString(2, person.getGender().toString());**

**//convert java.time.LocalDate to java.sql.Date**

**preparedStatement.setDate(3, java.sql.Date.*valueOf*(person.getBirthdate()));**

**preparedStatement.setString(4, person.getAddress());**

**preparedStatement.setLong(5, person.getMobile());**

**preparedStatement.setLong(6, person.getAdharCard());**

**int n = preparedStatement.executeUpdate();**

**if(n==0) {**

**throw new SQLException("Unable be up update the person data");**

**}**

**return "PERSON:"+ person.getAdharCard()+ " IS UPDATED TO DATABASE";**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**@Override**

**public String deletePerson(Long adharCard) throws SQLException {**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**PreparedStatement preparedStatement = connection.prepareCall(QueryMapper.*DELETE\_PERSON*);**

**){**

**preparedStatement.setLong(1, adharCard);**

**int n = preparedStatement.executeUpdate();**

**if(n==0) {**

**throw new SQLException("Invalid AdharCard");**

**}**

**return "PERSON:"+adharCard+ " DELETED FROM DATABASE";**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**@Override**

**public Person getPersonById(Long adharCard) throws SQLException {**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**PreparedStatement preparedStatement = connection.prepareCall(QueryMapper.*GET\_PERSON\_BY\_ID*);**

**){**

**preparedStatement.setLong(1, adharCard);**

**ResultSet resultSet = preparedStatement.executeQuery();**

**if(resultSet.next()) {**

**Person person = new Person();**

**populate(person,resultSet);**

**return person;**

**}else {**

**throw new SQLException("Invalid AdharCard Number");**

**}**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**@Override**

**public List<Person> getAllPersons() throws SQLException {**

**try(**

**Connection connection = OracleConnection.*getConnection*();**

**Statement statement = connection.createStatement();**

**){**

**ResultSet resultSet = statement.executeQuery(QueryMapper.*GET\_ALL\_PERSONS*);**

**List<Person> personList = new ArrayList<>();**

**while(resultSet.next()) {**

**Person person = new Person();**

**//copy contents of the current row of the resultSet object into person object**

**populate(person,resultSet);**

**//add person object to the list**

**personList.add(person);**

**}**

**return personList;**

**}catch(SQLException e) {**

**throw e;**

**}**

**}**

**private void populate(Person person, ResultSet resultSet) throws SQLException {**

**person.setAdharCard(resultSet.getLong("ADHAR\_CARD"));**

**person.setName(resultSet.getString("NAME"));**

**//convert String -> GenderTyp**

**person.setGender( GenderTyp.*valueOf*(resultSet.getString("GENDER")));**

**//convert java.sql.Date to java.time.LocalDate**

**person.setBirthdate(resultSet.getDate("BIRTHDATE").toLocalDate());**

**person.setAddress(resultSet.getString("ADDRESS"));**

**person.setMobile(resultSet.getLong("MOBILE"));**

**}**

**}**

**—------------------------------------------------------------------------------------------**

**JUNIT5**

**----------------------------------------------------------------------------------------------------**

**<dependency>**

**<groupId>org.junit.jupiter</groupId>**

**<artifactId>junit-jupiter-engine</artifactId>**

**<version>5.9.1</version>**

**<scope>test</scope>**

**</dependency>**

**<dependency>**

**<groupId>org.junit.jupiter</groupId>**

**<artifactId>junit-jupiter-params</artifactId>**

**<version>5.9.1</version>**

**<scope>test</scope>**

**</dependency>**

**package com.wipro.app;**

**import static org.junit.jupiter.api.Assertions.*assertEquals*;**

**import static org.junit.jupiter.api.Assertions.*assertTrue*;**

**import org.junit.jupiter.api.DisplayName;**

**import org.junit.jupiter.api.RepeatedTest;**

**import org.junit.jupiter.api.RepetitionInfo;**

**import org.junit.jupiter.api.TestInfo;**

**import org.junit.jupiter.params.ParameterizedTest;**

**import org.junit.jupiter.params.provider.ValueSource;**

**/\***

**\* Parameterized Testing & Repeated Testing**

**\* @ParamerizedTest and @RepeatedTest and sub types of Test**

**\***

**\* We cannot combine @Test with @ParamerizedTest or with @RepeatedTest**

**\*/**

**class AppTest {**

**@ParameterizedTest**

**@ValueSource(ints = {1, 3, 5, -3, 15, Integer.*MAX\_VALUE*}) // six numbers**

**void isOdd\_ShouldReturnTrueForOddNumbers(int number) {**

***assertTrue*(number%2 != 0);**

**}**

**@ParameterizedTest**

**@ValueSource(strings = {"cali","mali","infi","fruiti"})**

**void endsWith(String string) {**

***assertTrue*(string.endsWith("li"));**

**}**

**@DisplayName("Test Blanks")**

**@ParameterizedTest**

**@ValueSource(strings = {"", " "})**

**void isBlank\_ShouldReturnTrueForNullOrBlankStrings(String input) {**

***assertTrue*(input.isBlank());**

**}**

**@DisplayName("Repeated Test")**

**@RepeatedTest(value = 5)**

**void repeatedMethod(RepetitionInfo repInfo, TestInfo info ) {**

**// System.out.println(repInfo.getCurrentRepetition());**

**// System.out.println(repInfo.getTotalRepetitions());**

**System.*out*.println(info.getDisplayName());**

**int i=3;**

***assertEquals*(repInfo.getCurrentRepetition(),i);**

**}**

**}**

**Test Suite**

**To run multiple Test classes in a batch mode.**

**We need the following maven dependency:**

**<dependency>**

**<groupId>org.junit.platform</groupId>**

**<artifactId>junit-platform-suite-engine</artifactId>**

**<version>1.9.3</version>**

**<scope>test</scope>**

**</dependency>**

**—------------------------------------------------------------**

**package com.wipro.model;**

**import static org.junit.jupiter.api.Assertions.\*;**

**import java.time.LocalDate;**

**import org.junit.jupiter.api.AfterAll;**

**import org.junit.jupiter.api.AfterEach;**

**import org.junit.jupiter.api.BeforeAll;**

**import org.junit.jupiter.api.BeforeEach;**

**import org.junit.jupiter.api.Test;**

**class PersonTest {**

**private Person person=null;**

**@BeforeAll**

**static void setUpBeforeClass() throws Exception {**

**System.*out*.println("JUnit testing begins..");**

**}**

**@AfterAll**

**static void tearDownAfterClass() throws Exception {**

**System.*out*.println("End of junit testing");**

**}**

**@BeforeEach**

**void setUp() throws Exception {**

**person = new Person(987676767891L,"Smith",GenderTyp.*MALE*,LocalDate.*of*(1998, 10,10),"Hyderabad",8767543255L);**

**}**

**@AfterEach**

**void tearDown() throws Exception {**

**person=null;**

**}**

**@Test**

**void testPersonLongStringGenderTypLocalDateStringLong() {**

***assertEquals*(987676767891L, person.getAdharCard());**

**}**

**@Test**

**void testGetAdharCard() {**

***assertEquals*(987676767891L, person.getAdharCard());**

**}**

**@Test**

**void testSetAdharCardForTrue() {**

**person.setAdharCard(1L);**

***assertEquals*(1L,person.getAdharCard());**

**}**

**@Test**

**void testSetAdharCardForFalse() {**

**person.setAdharCard(1L);**

***assertNotEquals*(2L,person.getAdharCard());**

**}**

**}**

**—------------------------------------------------------------**

**package com.wipro.model;**

**import static org.junit.jupiter.api.Assertions.*assertEquals*;**

**import static org.junit.jupiter.api.Assertions.*fail*;**

**import java.time.LocalDate;**

**import org.junit.jupiter.api.Test;**

**public class PersonTest1 {**

**@Test**

**void testGetName() {**

**Person person = new Person();**

**person.setName("Clarke");**

***assertEquals*("Clarke",person.getName());**

**}**

**@Test**

**void testSetName() {**

**Person person = new Person();**

**person.setName("Clarke");**

***assertEquals*("Clarke",person.getName());**

**}**

**@Test**

**void testGetGender() {**

**Person person = new Person();**

**person.setGender(GenderTyp.*MALE*);**

***assertEquals*(GenderTyp.*MALE*,person.getGender());**

**}**

**@Test**

**void testSetGender() {**

**Person person = new Person();**

**person.setGender(GenderTyp.*MALE*);**

***assertEquals*(GenderTyp.*MALE*,person.getGender());**

**}**

**@Test**

**void testGetBirthdate() {**

**Person person = new Person(987676767891L,"Smith",GenderTyp.*MALE*,LocalDate.*of*(1998, 10,10),"Hyderabad",8767543255L);**

***assertEquals*(LocalDate.*of*(1998, 1, 10), person.getBirthdate());**

**}**

**@Test**

**void testSetBirthdate() {**

**Person person = new Person(987676767891L,"Smith",GenderTyp.*MALE*,LocalDate.*of*(1998, 10,10),"Hyderabad",8767543255L);**

**person.setBirthdate(LocalDate.*of*(2001, 1,1));**

***assertEquals*(LocalDate.*of*(2001, 1, 1), person.getBirthdate());**

**}**

**}**

**—-------------------------------------------------**

**package com.wipro.model;**

**import static org.junit.jupiter.api.Assertions.*assertEquals*;**

**import static org.junit.jupiter.api.Assertions.*assertNotEquals*;**

**import static org.junit.jupiter.api.Assertions.*fail*;**

**import java.time.LocalDate;**

**import org.junit.jupiter.api.Test;**

**public class PersonTest2 {**

**@Test**

**void testGetAddress() {**

**Person person = new Person(987676767891L,"Smith",GenderTyp.*MALE*,LocalDate.*of*(1998, 10,10),"Hyderabad",8767543255L);**

***assertEquals*("Hyderabad", person.getAddress());**

**}**

**@Test**

**void testSetAddress() {**

**Person person = new Person(987676767891L,"Smith",GenderTyp.*MALE*,LocalDate.*of*(1998, 10,10),"Hyderabad",8767543255L);**

**person.setAddress("Chennai");**

***assertEquals*("Chennai", person.getAddress());**

**}**

**@Test**

**void testGetMobile() {**

**Person person= new Person();**

**person.setMobile(8767898999L);**

***assertNotEquals*(9878999989L,person.getMobile());**

**}**

**@Test**

**void testSetMobile() {**

**Person person = new Person(987676767891L,"Smith",GenderTyp.*MALE*,LocalDate.*of*(1998, 10,10),"Hyderabad",8767543255L);**

**person.setMobile(9759871239L);**

***assertEquals*(9759871239L,person.getMobile());**

**}**

**}**

**—-----------------------------------------------------------**

**package com.wipro.suite;**

**import org.junit.platform.suite.api.SelectClasses;**

**import org.junit.platform.suite.api.SelectPackages;**

**import org.junit.platform.suite.api.Suite;**

**@Suite**

**//@SelectPackages("com.wipro.model")**

**@SelectClasses({com.wipro.model.PersonTest.class,**

**com.wipro.app.AppTest.class})**

**public class PersonSuite {**

**}**

**—-----------------------------------------------------------------**

### **Common Assertions in JUnit 5**

* assertEquals (expected, actual): Asserts that the expected and actual values are equal.
* assertNotEquals(expected, actual): Asserts that the expected values and the actual values are not equal.
* assertTrue(condition): This asserts whether the given condition is true. Test case passes if it’s true and fails if not.
* assertFalse(condition): This asserts whether the given condition is false. Test case passes if it’s false and fails if not.
* assertNull(value): This asserts whether the given value is null. Test case passes if it’s null and fails if not.
* assertNotNull(value): This asserts whether the given value is not null. Test case passes if it’s not null and fails if not.
* assertArrayEquals(expectedArray, actualArray): This asserts whether the expected and actual arrays are equal. Test case passes if they’re equal and fails if not.
* assertSame(expected, actual): This asserts whether the expected and actual references point to the same object. Test case passes if it’s true and fails if not.
* assertNotSame(expected, actual): This asserts whether the expected and actual references do not point to the same object. Test case passes if it’s true and fails if not.
* assertThrows(exceptionType, executable): This asserts whether the executable throws an exception of the specified type. Test case passes if it throws an exception and fails if not.

**Assignment:**

**Create Test class for a class that contains methods that perform basic arithmetic operations of two numbers.**

**public class Calculator{**

**public int add(int x, int y){**

**return x+y;**

**}**

**// other methods**

**}**

**public class CalculatorTest{**

**//Test methods**

**}**

**—-----------------------------------------------------------------------------------------------------------**

**Logger Frameworks**

Logging is writing the state of a program at various stages of its

execution to some repository such as a log file.

By logging, simple yet explanatory statements can be sent to text

file, console, or any other repository.

Using logging, a reliable monitoring and debugging solution can be

Achieved

Logging in Java requires using one or more logging frameworks. These frameworks provide the objects, methods, and configuration necessary to create and send log messages.

Java provides a built-in framework in the [java.util.logging](https://docs.oracle.com/en/java/javase/12/docs/api/java.logging/java/util/logging/package-summary.html) package. There are also many third-party frameworks, including [Log4j](https://logging.apache.org/log4j/2.x/), [Logback](https://logback.qos.ch/), and [tinylog](https://www.tinylog.org/).

You can also use an abstraction layer, such as [SLF4J](https://www.slf4j.org/) and [Apache Commons Logging](https://commons.apache.org/proper/commons-logging/), which decouples your code from the underlying logging framework so you can switch between logging frameworks on the fly.

Apache Log4j is a versatile, industrial-grade Java logging framework composed of an API, its implementation, and components to assist the deployment for various use cases.

Log4j comprises of three main components:

Logger

Appender

Layout

—---------------------------------------------------------------------------------------

Log4J PatternLayout conversion specifiers

<https://www.tutorialspoint.com/log4j/log4j_patternlayout.htm>

<dependency>

<groupId>log4j</groupId>

<artifactId>log4j</artifactId>

<version>1.2.17</version>

</dependency>

—----------------------------------------log4j.xml—----------------------------

1.Create a file, log4j.xml in src/main/resources folder

2. Create a folder called, log. Right-click on project name, new->folder

<?xml version=*"1.0"* encoding=*"UTF-8"* ?>

<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">

<log4j:configuration debug=*"true"*

xmlns:log4j=*'http://jakarta.apache.org/log4j/'*>

<appender name=*"CONSOLE-APPENDER"* class=*"org.apache.log4j.ConsoleAppender"*>

<layout class=*"org.apache.log4j.PatternLayout"*>

<param name=*"ConversionPattern"*

value=*"%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n"* />

</layout>

</appender>

<appender name=*"FILE-APPENDER"* class=*"org.apache.log4j.RollingFileAppender"*>

<param name=*"append"* value=*"false"* />

<param name=*"maxFileSize"* value=*"10MB"* />

<param name=*"maxBackupIndex"* value=*"2"* />

<param name=*"file"* value=*"./log/person.html"* />

<layout class=*"org.apache.log4j.HTMLLayout"*>

<param name=*"ConversionPattern"*

value=*"%d{yyyy-MM-dd HH:mm:ss} %-5p %c{1}:%L - %m%n"* />

</layout>

</appender>

<root>

<level value=*"DEBUG"* />

<appender-ref ref=*"CONSOLE-APPENDER"* />

<appender-ref ref=*"FILE-APPENDER"* />

</root>

</log4j:configuration>

App.java

**package** com.wipro.app;

**import** java.time.LocalDate;

**import** java.util.List;

**import** java.util.Scanner;

**import** org.apache.log4j.Logger;

**import** com.wipro.exception.PersonException;

**import** com.wipro.model.GenderTyp;

**import** com.wipro.model.Person;

**import** com.wipro.service.PersonService;

**import** com.wipro.service.PersonServiceImpl;

**public** **class** App {

**private** **static** PersonService *personService* = **new** PersonServiceImpl();

**private** **static** Scanner *scanner* = **new** Scanner(System.***in***);

**private** **static** Logger *logger* = Logger.*getLogger*(App.**class**);

**public** **static** **void** main( String[] args ){

**try** {

Person person = **new** Person(787567487688L,"Venkat",

GenderTyp.***MALE***,LocalDate.*of*(2005, 10, 15),"New Delhi",8693265789L);

String message = *personService*.addPerson(person);

System.***out***.println(message);

*logger*.info(message);

}**catch**(PersonException e) {

// e.printStackTrace();

*logger*.error(e.getMessage(), e);

}

**try** {

List<Person> personList = *personService*.getAllPersons();

personList.forEach(System.***out***::println);

*logger*.info("Displaying details all persons..");

}**catch**(PersonException e) {

// e.printStackTrace();

*logger*.error(e.getMessage(),e);

}

// try {

// System.out.println("Enter adharcard: ");

// long adharCard = Long.parseLong(scanner.nextLine());

// Person person = personService.getPersonById(adharCard);

// System.out.println(person);

//

// }catch(PersonException e) {

// e.printStackTrace();

// }

// try {

// System.out.println("Enter adharcard of the person to delete: ");

// long adharCard = Long.parseLong(scanner.nextLine());

// String message = personService.deletePerson(adharCard);

// System.out.println(message);

//

// }catch(PersonException e) {

// e.printStackTrace();

// }

// try {

// System.out.println("Enter adharcard of the person to update: ");

// long adharCard = Long.parseLong(scanner.nextLine());

// Person person = personService.getPersonById(adharCard);

//

// System.out.println("Hi, "+ person.getName()+ "Do want to change your name(yes/no)");

// String option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter your new name: ");

// person.setName(scanner.nextLine());

// }

//

// System.out.println("Change Gender? (yes/no): ");

// option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter Gender ");

// person.setGender( GenderTyp.valueOf(scanner.nextLine()));

// }

//

//

// System.out.println("Change Birthdate? (yes/no): ");

// option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter New Birthdate(yyyy-mm-dd) ");

// //String -> LocalDate

// person.setBirthdate(LocalDate.parse(scanner.nextLine()));

// }

//

//

// System.out.println("Change Address? (yes/no): ");

// option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter New Address: ");

// //String -> LocalDate

// person.setAddress(scanner.nextLine());

// }

//

//

// System.out.println("Change Mobile? (yes/no): ");

// option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter New Mobile N umber: ");

// person.setMobile(Long.parseLong(scanner.nextLine()));

// }

//

// String message = personService.updatePerson(person);

// System.out.println(message);

//

// }catch(PersonException e) {

// e.printStackTrace();

// }

}

}

—------------------------------VisualVM—-------------------------------------

<https://www.geeksforgeeks.org/how-to-install-visualvm-in-linux/>

Install from Terminal

$ sudo [apt](https://www.geeksforgeeks.org/apt-command-in-linux-with-examples/) update

$ sudo apt install visualvm

Open the Applications, click on the VisualVM icon. The VisualVM window appears.

—---------------------------------------------------------------------------------

package com.wipro.app;

import java.time.LocalDate;

import java.util.List;

import java.util.Scanner;

import org.apache.log4j.Logger;

import com.wipro.exception.PersonException;

import com.wipro.model.GenderTyp;

import com.wipro.model.Person;

import com.wipro.service.PersonService;

import com.wipro.service.PersonServiceImpl;

public class App {

private static PersonService personService = new PersonServiceImpl();

private static Scanner scanner = new Scanner(System.in);

private static Logger logger = Logger.getLogger(App.class);

public static void main( String[] args ){

try {

Person person = new Person(123123123898L,"Manisha",

GenderTyp.FEMALE,LocalDate.of(1999, 11, 18),"Kanput",7693265700L);

String message = personService.addPerson(person);

System.out.println(message);

logger.info(message);

}catch(PersonException e) {

// e.printStackTrace();

logger.error(e.getMessage(), e);

}

try {

List<Person> personList = personService.getAllPersons();

personList.forEach(System.out::println);

logger.info("Displaying details all persons..");

}catch(PersonException e) {

// e.printStackTrace();

logger.error(e.getMessage(),e);

}

while(true) {

try {

System.out.println("Enter adharcard: ");

long adharCard = Long.parseLong(scanner.nextLine());

Person person = personService.getPersonById(adharCard);

System.out.println(person);

}catch(PersonException e) {

e.printStackTrace();

}

System.out.println("Press X to stop");

if(scanner.nextLine().equalsIgnoreCase("X")) break;

}

// try {

// System.out.println("Enter adharcard of the person to delete: ");

// long adharCard = Long.parseLong(scanner.nextLine());

// String message = personService.deletePerson(adharCard);

// System.out.println(message);

//

// }catch(PersonException e) {

// e.printStackTrace();

// }

// try {

// System.out.println("Enter adharcard of the person to update: ");

// long adharCard = Long.parseLong(scanner.nextLine());

// Person person = personService.getPersonById(adharCard);

//

// System.out.println("Hi, "+ person.getName()+ "Do want to change your name(yes/no)");

// String option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter your new name: ");

// person.setName(scanner.nextLine());

// }

//

// System.out.println("Change Gender? (yes/no): ");

// option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter Gender ");

// person.setGender( GenderTyp.valueOf(scanner.nextLine()));

// }

//

//

// System.out.println("Change Birthdate? (yes/no): ");

// option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter New Birthdate(yyyy-mm-dd) ");

// //String -> LocalDate

// person.setBirthdate(LocalDate.parse(scanner.nextLine()));

// }

//

//

// System.out.println("Change Address? (yes/no): ");

// option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter New Address: ");

// //String -> LocalDate

// person.setAddress(scanner.nextLine());

// }

//

//

// System.out.println("Change Mobile? (yes/no): ");

// option = scanner.nextLine();

// if(option.equalsIgnoreCase("yes")) {

// System.out.println("Enter New Mobile N umber: ");

// person.setMobile(Long.parseLong(scanner.nextLine()));

// }

//

// String message = personService.updatePerson(person);

// System.out.println(message);

//

// }catch(PersonException e) {

// e.printStackTrace();

// }

}

}

—----------------------------Running Java In Debug Mode—--------

The most commonly used debug options are:

* ***Step Into (F5) –* This operation goes inside the methods** used in the current line (if any); else, it proceeds to the next line. In this example, it will take the debugger inside the method *isPerfectSquare()*
* ***Step Over (F6) –* This operation processes the current line and proceeds to the next line.** In this example, this will execute the method *isPerfectSquare()* and proceed to the next line
* ***Step Return (F7) –* This operation finishes the current method and takes us back to the calling method.** Since in this case, we have a breakpoint in the loop, it will be still within the method, else it would go back to the main method
* ***Resume (F8) –* This operation will simply continue with the execution until the program ends** unless we hit any further breakpoint

**package** com.wipro.app;

**public** **class** PerfectSquareCounter {

**private** **static** **int** *evenPerfectSquareNumbers* = 0;

**public** **static** **void** main(String[] args) {

**int** i = 100;

System.***out***.println("Total Perfect Squares: " + *calculateCount*(i));

System.***out***.println("Even Perfect Squares : " + *evenPerfectSquareNumbers*);

}

**public** **static** **int** calculateCount(**int** i) {

**int** perfectSquaresCount = 0;

**for** (**int** number = 1; number <= i; number++) {

**if** (*isPerfectSquare*(number)) {

perfectSquaresCount++;

**if** (number % 2 == 0) {

*evenPerfectSquareNumbers*++;

}

}

}

**return** perfectSquaresCount;

}

**private** **static** **boolean** isPerfectSquare(**int** number) {

**double** sqrt = Math.*sqrt*(number);

**return** sqrt - Math.*floor*(sqrt) == 0;

}

}

======================================================================================================================

User Interface

Front-End Technologies:

HTML5 , CSS3, Javascript, Bootstrap, React JS

**HTML5**

### **Introduction to HTML/CSS/Javascript**

If you are the content provider, read HTML. If you are the graphic designer, read CSS. If you are a programmer and want to add dynamic effects to your web page, read JavaScript. But if you operate in OMO (one-man-operated) and are expected to create a reasonably good-looking website, you need to understand HTML, CSS and JavaScript

## **What is HTML?**

* HTML stands for Hyper Text Markup Language
* HTML is the standard markup language for creating Web pages
* HTML describes the structure of a Web page
* HTML consists of a series of elements
* HTML elements tell the browser how to display the content

**HTML5 Structure:**

**<!DOCTYPE html>**

**<html>**

**<head>**

**</head>**

**<body>**

**<h1>Welcome To HTML5 </h1>**

**</body>**

**</html>**

—-----------------------------------------------

**HTML Element**

The HTML element is everything from the start tag to the end tag:

**<tagname>Content goes here...</tagname>**

**Examples of some HTML elements:**

<h1 align=’center’>My First Heading</h1>

Here align is attribute of h1

<p>My first paragraph</p>

**—-----------------------------------------------------------------**

**HTML DOM (Document Object Model)**

**The HTML DOM is an Object Model for HTML. It defines:**

* HTML elements as objects
* Properties for all HTML elements
* Methods for all HTML elements
* Events for all HTML elements

**The HTML DOM is an API (Programming Interface) for JavaScript:**

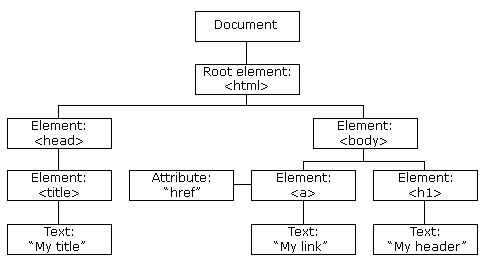
* JavaScript can add/change/remove HTML elements
* JavaScript can add/change/remove HTML attributes
* JavaScript can add/change/remove CSS styles
* JavaScript can react to HTML events
* JavaScript can add/change/remove HTML events

When a web page is loaded, the browser creates a **D**ocument **O**bject **M**odel of the page.

The **HTML DOM** model is constructed as a tree of **Objects**:

### 

### **The HTML DOM Tree of Objects**

****

## **Web Browsers**

The purpose of a web browser (Chrome, Edge, Firefox, Safari) is to read HTML documents and display them correctly.

A browser does not display the HTML tags, but uses them to determine how to display the document:

## **HTML Headings**

HTML headings are defined with the <h1> to <h6> tags.

<h1> defines the most important heading. <h6> defines the least important heading:

## **HTML Links**

HTML links are defined with the <a> tag:

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>My HTML5 Page </title>**

**</head>**

**<body>**

**<h2>HTML Links</h2>**

**<p>HTML links are defined with the a tag:</p>**

**<a href="**[**https://www.google.com**](https://www.google.com)**">Google Home Page</a>**

**</body>**

**</html>**

**HTML Images**

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h2>HTML Images</h2>**

**<p>HTML images are defined with the img tag:</p>**

**<img src="w3schools.jpg" alt="W3Schools.com" width="104" height="142">**

**</body>**

**</html>**

## **Empty HTML Elements**

HTML elements with no content are called empty elements.

The <br> tag defines a line break, and is an empty element without a closing tag:

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p>This is a <br> paragraph with a <br>line break.</p>**

**</body>**

**</html>**

## **HTML Attributes**

* All HTML elements can have attributes
* Attributes provide additional information about elements
* Attributes are always specified in the start tag
* Attributes usually come in name/value pairs like: name="value"

<img src="img\_girl.jpg" width="500" height="600">

C:\

images html

rose.jpg home.html

Absolute Path

<img src=’c:\images\rose.jpg’ </img>

Relative Path

<img src=’ ..\images\rose.jpg’ </img>

## **The style Attribute**

The style attribute is used to add styles to an element, such as color, font, size, and more.

<p style="color:red;">This is a red paragraph.</p>

Note: Not recommended

Note:

Double quotes around attribute values are the most common in HTML, but single quotes can also be used.

In some situations, when the attribute value itself contains double quotes, it is necessary to use single quotes:

<p title='John "ShotGun" Nelson'>

## **HTML Horizontal Rules**

The <hr> stands for **horizontal ruler** tag defines a thematic break in an HTML page, and is most often displayed as a horizontal rule.

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h1>This is heading 1</h1>**

**<p>This is some text.</p>**

**<hr>**

**<h2>This is heading 2</h2>**

**<p>This is some other text.</p>**

**<hr>**

**<h2>This is heading 2</h2>**

**<p>This is some other text.</p>**

**</body>**

**</html>**

## **The HTML Style Attribute**

Setting the style of an HTML element, can be done with the style attribute.

The HTML style attribute has the following syntax:

<*tagname* style="*property*:*value;*">

The ***property*** is a CSS property. The ***value*** is a CSS value.

**<body style="background-color:powderblue;">**

**<h1>This is a heading</h1>**

**<p>This is a paragraph.</p>**

**</body>**

**HTML Formatting Elements**

**<!DOCTYPE html>**

**<html>**

**<body>**

**<p><b>This text is bold</b></p>**

**<p><i>This text is italic</i></p>**

**<p>This is<sub> subscript</sub> and <sup>superscript</sup></p>**

**</body>**

**</html>**

## **HTML Comment Tag**

You can add comments to your HTML source by using the following syntax:

**<!-- Write your comments here →**

**—--------------------------------------------------------------------------------------**

[**https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTML\_CSS\_Basics.html**](https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/HTML_CSS_Basics.html)

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>My HTML5 Document</title>**

**</head>**

**<body>**

**<header> <h2><font color="green">Welcome To My HTML5 Demo Page</font></h2></header>**

**<hr>**

**<section>**

**Understanding basics of HTML5<br>**

**ThanQ**

**</section>**

**<hr>**

**<footer><h5 align="center">Wipro 2024</h5></footer>**

**</body>**

**</html>**

##### **Most Frequently-Used HTML Elements**

The most frequently-used HTML elements are:

* block elements: <p> (paragraph), <br> (line break), <h1> to <h6> (heading level 1 to 6), <hr> (horizontal rule), <ul><li> (unordered list), <ol><li> (ordered list), <div>, <header>,<footer>,

<section>

* inline elements: <b> (bold), <i> (italic), <img> (image), <a> (anchor for hyperlink).
* table <table><tr><th><td>.
* Forms:

—----------------------------------------------------------------------------------------------

**<!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="utf-8">**

**<title>Table and Images</title>**

**<style>**

**table { /\* table \*/**

**border: 1px solid black;**

**border-spacing: 5px;**

**border-collapse: separate;**

**}**

**th, td { /\* cells \*/**

**border: 1px solid #aaa;**

**padding: 5px 10px;**

**}**

**</style>**

**</head>**

**<body>**

**<h1>Table and Images</h1>**

**<table>**

**<caption>Logo of Languages</caption>**

**<tr>**

**<th>S/No</th>**

**<th>Language</th>**

**<th>Logo</th>**

**</tr>**

**<tr>**

**<td>1.</td>**

**<td>HTML5</td>**

**<td><img src="../images/html5-logo.png" alt="HTML Logo" height="64" width="64"></td>**

**</tr>**

**<tr>**

**<td>2.</td>**

**<td>CSS3</td>**

**<td><img src="../images/css3-logo.png" alt="CSS Logo" height="64" width="64"></td>**

**</tr>**

**<tr>**

**<td>3.</td>**

**<td>JavaScript</td>**

**<td><img src="../images/js-logo.png" alt="JavaScript Logo" height="64" width="64"></td>**

**</tr>**

**</table>**

**</body>**

**</html>**

##### **Division <div>...</div> and Span <span>...</span>**

<!DOCTYPE html>

<html>

<head>

<style>

.myDiv {

border: 5px outset red;

background-color: lightblue;

text-align: center;

}

</style>

</head>

<body>

<div class="myDiv">

<h2 style="color:green;">This is a heading in a div element</h2>

<p>This is <span style ="color:red;font-size:40px">Inline styling</span> in a div element.</p> </div>

</body>

</html>

##### **Header <header>...</header>, Footer <footer>...</footer> and Section <section>...</section>**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**.myDiv {**

**border: 5px outset red;**

**background-color: lightblue;**

**text-align: center;**

**}**

**</style>**

**</head>**

**<body>**

**<header>HTML5 New Semantic Blocks</header>**

**<section class="myDiv">**

**<h2 style="color:green;">This is a heading in a div element</h2>**

**<p>This is <span style ="color:red;font-size:40px">Inline styling</span>**

**in a div element.</p>**

**</section>**

**<!-- <div class="myDiv">**

**<h2 style="color:green;">This is a heading in a div element</h2>**

**<p>This is <span style ="color:red;font-size:40px">Inline styling</span>**

**in a div element.</p>**

**</div> -->**

**<footer>Copyright 2024@Wipro</footer>**

**</body>**

**</html>**

#### **HTML Tables**

Table-related tags are meant for tabulating data. (Older HTML documents tend to use <table> for formatting the document to divide the document into columns/sections, which should be avoided. Use style sheet for formatting instead.)

The basic unit of a table is a *cell*. Cells are grouped into *row*. Rows are grouped to form the *table*. This corresponds well to the "row-centric" approach in the display.

The essential tags used by table are:

* <table>...</table>: contains the entire table.
* <tr>...</tr>: contains a row.
* <th>...</th> and <td>...</td>: contain a *header* cell and a *data* (*detail*) cell respectively.

Additional tags are:

* <caption>...</caption>: specifies a caption.
* <thead>...</thead>, <tbody>...</tbody>, and <tfoot>...</tfoot>: for marking out the table header, body and footer.
* <colgroup>...</colgroup> and <col>...</col>: for applying styles to column group and column respectively.

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**thead,tfoot {**

**background-color: #2c5e77;**

**color: #fff;**

**}**

**tbody {**

**background-color: #e4f0f5;**

**}**

**table {**

**border-collapse: collapse;**

**border: 2px solid rgb(140 140 140);**

**font-family: sans-serif;**

**font-size: 0.8rem;**

**letter-spacing: 1px;**

**}**

**caption {**

**caption-side: bottom;**

**padding: 10px;**

**}**

**th, td {**

**border: 1px solid rgb(160 160 160);**

**padding: 8px 10px;**

**}**

**td {**

**text-align: center;**

**}**

**</style>**

**</head>**

**<body>**

**<table>**

**<caption>**

**Council budget (in $) 2024**

**</caption>**

**<thead>**

**<tr>**

**<th>Items</th>**

**<th>Expenditure</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr>**

**<th>Donuts</th>**

**<td>3,000</td>**

**</tr>**

**<tr>**

**<th>Stationery</th>**

**<td>18,000</td>**

**</tr>**

**</tbody>**

**<tfoot>**

**<tr>**

**<th>Totals</th>**

**<td>21,000</td>**

**</tr>**

**</tfoot>**

**</table>**

**</body>**

**</html>**

**HTML Forms**

[**https://www.almabetter.com/bytes/tutorials/html/html5-form**](https://www.almabetter.com/bytes/tutorials/html/html5-form)

**<!DOCTYPE html>**

**<html>**

**<head>**

**<title>Login Form</title>**

**<style>**

**thead,tfoot {**

**background-color: #2c5e77;**

**color: #fff;**

**}**

**tbody {**

**background-color: #e4f0f5;**

**}**

**table {**

**border-collapse: collapse;**

**border: 2px solid rgb(140 140 140);**

**font-family: sans-serif;**

**font-size: 0.8rem;**

**letter-spacing: 1px;**

**}**

**caption {**

**caption-side: bottom;**

**padding: 10px;**

**}**

**th, td {**

**border: 1px solid rgb(160 160 160);**

**padding: 8px 10px;**

**}**

**td {**

**text-align: center;**

**}**

**</style>**

**</head>**

**<body>**

**<header>**

**<h3 align="center">Login Form</h3>**

**</header>**

**<section >**

**<form id="frmId" action="#" method="GET"></form>**

**<table align="center">**

**<thead>**

**<tr>**

**<th colspan="2">Enter Credentials</th>**

**</tr>**

**</thead>**

**<tbody>**

**<tr>**

**<th><label for="username">UserName</label></th>**

**<td><input type="text" id= "username" size="30" /></td>**

**</tr>**

**<tr>**

**<th><label for="password">Password</label></th>**

**<td><input type="password" id= "password" size="30" /></td>**

**</tr>**

**<tr>**

**<td colspan="2"><input type="submit" id= "submit" size="30"/> </td>**

**</tr>**

**</tbody>**

**</table>**

**</form>**

**</section>**

**<footer>**

**<h5 align="center">Copyright 2024@ Wipro Pvt. Ltd.</h5>**

**</footer>**

**</body>**

**</html>**

#### **Anchors and Hyperlinks**

##### **<a href="*url*" target="...">...</a>**

**target="\_blank": opens the linked document in a new tab or window.**

**target="\_self" (default): opens the linked document in the same window/frame.**

**target="\_parent": opens the linked document in the parent frame.**

**target="\_top": opens the linked document in the full body of the window.**

**target="*frame-name*": opens the linked document in the named frame.**

##### **URLs (Uniform Resource Locators)**

**A URL uniquely identifies a piece of resource over the Internet. A URL is made up of 4 parts as follows:**

***protocol*://*hostname*:*port*/*path\_and\_filename***

1. **Protocol: e.g., http, ftp, mailto, file, telnet and others.**
2. **Server's domain name (e.g., www.w3c.org) or IP address (e.g., 127.0.0.1). The DNS (Domain Name Service) translates a domain name to an IP address.**
3. **Port number (optional): the TCP port number on which the server application is running. The default TCP port number is used if port number is omitted from the URL. For example, default TCP port number 80 will be used for HTTP, 21 for FTP.**
4. **Directory path and file name: Unix-style forward-slash '/' is used as the path separator (instead of Windows-style back-slash '\'). Directory path and filename of the URL are case sensitive.**

## **What is HTML Web Storage?**

**With web storage, web applications can store data locally within the user's browser.**

**Before HTML5, application data had to be stored in cookies, included in every server request. Web storage is more secure, and large amounts of data can be stored locally, without affecting website performance.**

**Unlike cookies, the storage limit is far larger (at least 5MB) and information is never transferred to the server.**

## **The localStorage Object**

**The localStorage object stores the data with no expiration date. The data will not be deleted when the browser is closed, and will be available the next day, week, or year.**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function clickCounter() {**

**if (typeof(Storage) !== "undefined") {**

**if (localStorage.clickcount) {**

**localStorage.clickcount = Number(localStorage.clickcount)+1;**

**} else {**

**localStorage.clickcount = 1;**

**}**

**document.getElementById("result").innerHTML = "You have clicked the button " + localStorage.clickcount + " time(s).";**

**} else {**

**document.getElementById("result").innerHTML = "Sorry, your browser does not support web storage...";**

**}**

**}**

**</script>**

**</head>**

**<body>**

**<p><button onclick="clickCounter()" type="button">Click me!</button></p>**

**<div id="result"></div>**

**<p>Click the button to see the counter increase.</p>**

**<p>Close the browser tab (or window), and try again, and the counter will continue to count (is not reset).</p>**

**</body>**

**</html>**

## **The sessionStorage Object**

**The sessionStorage object is equal to the localStorage object, except that it stores the data for only one session. The data is deleted when the user closes the specific browser tab.**

**localStorage example**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function clickCounter() {**

**if (typeof(Storage) !== "undefined") {**

**if (localStorage.clickcount) {**

**localStorage.clickcount = Number(localStorage.clickcount)+1;**

**} else {**

**localStorage.clickcount = 1;**

**}**

**document.getElementById("result").innerHTML = "You have clicked the button " + localStorage.clickcount + " time(s).";**

**} else {**

**document.getElementById("result").innerHTML = "Sorry, your browser does not support web storage...";**

**}**

**}**

**</script>**

**</head>**

**<body>**

**<p><button onclick="clickCounter()" type="button">Click me!</button></p>**

**<div id="result"></div>**

**<p>Click the button to see the counter increase.</p>**

**<p>Close the browser tab (or window), and try again, and the counter will continue to count (is not reset).</p>**

**</body>**

**</html>**

**sessionStorage Example**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function clickCounter() {**

**if (typeof(Storage) !== "undefined") {**

**if (sessionStorage.clickcount) {**

**sessionStorage.clickcount = Number(sessionStorage.clickcount)+1;**

**} else {**

**sessionStorage.clickcount = 1;**

**}**

**document.getElementById("result").innerHTML = "You have clicked the button " + sessionStorage.clickcount + " time(s) in this session.";**

**} else {**

**document.getElementById("result").innerHTML = "Sorry, your browser does not support web storage...";**

**}**

**}**

**</script>**

**</head>**

**<body>**

**<p><button onclick="clickCounter()" type="button">Click me!</button></p>**

**<div id="result"></div>**

**<p>Click the button to see the counter increase.</p>**

**<p>Close the browser tab (or window), and try again, and the counter is reset.</p>**

**</body>**

**</html>**

**—------------------------------------------------------------------------------------------------------------**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<script>**

**function clickCounter() {**

**if (typeof(Storage) !== "undefined") {**

**if (localStorage.clickcount) {**

**localStorage.clickcount = Number(localStorage.clickcount)+1;**

**} else {**

**localStorage.clickcount = 1;**

**}**

**document.getElementById("result").innerHTML = "You have clicked the button " + localStorage.clickcount + " time(s).";**

**} else {**

**document.getElementById("result").innerHTML = "Sorry, your browser does not support web storage...";**

**}**

**}**

**function resetCounter() {**

**if (typeof(Storage) !== "undefined") {**

**localStorage.setItem("clickcount","0");**

**} else {**

**document.getElementById("result").innerHTML = "Sorry, your browser does not support web storage...";**

**}**

**}**

**</script>**

**</head>**

**<body>**

**<p><button onclick="clickCounter()" type="button">Click me!</button></p>**

**<p><button onclick="resetCounter()" type="button">Reset</button></p>**

**<div id="result"></div>**

**<p>Click the button to see the counter increase.</p>**

**<p>Close the browser tab (or window), and try again, and the counter will continue to count (is not reset).</p>**

**</body>**

**</html>**

**—-------------------------------------------------------------------------**

# **HTML <nav> Tag**

**The <nav> tag defines a set of navigation links.**

**Notice that NOT all links of a document should be inside a <nav> element. The <nav> element is intended only for major blocks of navigation links.**

**<!DOCTYPE html>**

**<html>**

**<body>**

**<h1>The nav element</h1>**

**<p>The nav element defines a set of navigation links:</p>**

**<nav>**

**<a href="/html/">HTML</a> |**

**<a href="/css/">CSS</a> |**

**<a href="/js/">JavaScript</a> |**

**<a href="/python/">Python</a>**

**</nav>**

**</body>**

**</html>**

**—------------------------------------------------------------------------------------------------**

**CSS3**

**Cascading Style Sheet**

**Note:**

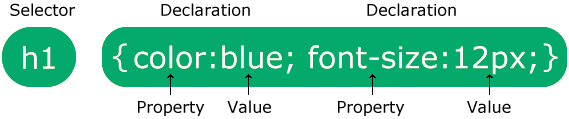
**A CSS comment is placed inside the <style> element, and starts with /\* and ends with \*/:**

##### **What is a Style Sheet?**

A *Style Sheet* is *a collection of style rules* that can be applied to *a selected set of HTML elements*. A style rule is used to control the appearance of HTML elements such as their font properties (e.g., type face, size and weight), color properties (e.g., background and foreground colors), alignment, margin, border, padding, and positioning.

The word *cascading* means that multiple style rules can be applied to the same HTML element. The browser follows a certain *cascading order* in finalizing a style to format the HTML element in a predictable fashion.

CSS Syntax:



We can divide CSS selectors into five categories:

* Simple selectors (select elements based on name, id, class)
* [Combinator selectors](https://www.w3schools.com/css/css_combinators.asp) (select elements based on a specific relationship between them)
* [Pseudo-class selectors](https://www.w3schools.com/css/css_pseudo_classes.asp) (select elements based on a certain state)
* [Pseudo-elements selectors](https://www.w3schools.com/css/css_pseudo_elements.asp) (select and style a part of an element)
* [Attribute selectors](https://www.w3schools.com/css/css_attribute_selectors.asp) (select elements based on an attribute or attribute value)

## **The CSS element Selector**

The element selector selects HTML elements based on the element name.

<!DOCTYPE html>

<html>

<head>

**<style>**

**p {**

**text-align: center;**

**color: red;**

**}**

**</style>**

</head>

<body>

<p>Every paragraph will be affected by the style.</p>

<p id="para1">Me too!</p>

<p>And me!</p>

</body>

</html>

## **2. The CSS id Selector**

The id selector uses the id attribute of an HTML element to select a specific element.

The id of an element is unique within a page, so the id selector is used to select one unique element!

To select an element with a specific id, write a hash (#) character, followed by the id of the element.

<!DOCTYPE html>

<html>

<head>

**<style>**

**#para1 {**

**text-align: center;**

**color: red;**

**}**

**</style>**

</head>

<body>

**<p id="para1">Hello World!</p>**

<p>This paragraph is not affected by the style.</p>

</body>

</html>

3. **The CSS class Selector**

The class selector selects HTML elements with a specific class attribute.

To select elements with a specific class, write a period (.) character, followed by the class name.

<!DOCTYPE html>

<html>

<head>

**<style>**

**.center {**

**text-align: center;**

**color: red;**

**}**

**</style>**

</head>

<body>

**<h1 class="center">Red and center-aligned heading</h1>**

**<p class="center">Red and center-aligned paragraph.</p>**

</body>

</html>

4. **The CSS Universal Selector**

The universal selector (\*) selects all HTML elements on the page.

<!DOCTYPE html>

<html>

<head>

**<style>**

**\* {**

**text-align: center;**

**color: blue;**

**}**

**</style>**

</head>

<body>

<h1>Hello world!</h1>

<p>Every element on the page will be affected by the style.</p>

<p id="para1">Me too!</p>

<p>And me!</p>

</body>

</html>

## **The CSS Grouping Selector**

The grouping selector selects all the HTML elements with the same style definitions.

**h1, h2, p {**

**text-align: center;**

**color: red;**

**}**

| [***element.class***](https://www.w3schools.com/cssref/sel_element_class.asp) | **p.intro** | **Selects only <p> elements with class="intro"** |
| --- | --- | --- |

**p.intro {**

**text-align: center;**

**color: red;**

**}**

<!DOCTYPE html>

<html>

<head>

**<style>**

**p.intro {**

**text-align: center;**

**color: blue;**

**}**

**</style>**

</head>

<body>

<h1>Hello world!</h1>

**<p class=”intro”>Every element on the page will be affected by the style.</p>**

<p id="para1">Me too!</p>

<p>And me!</p>

</body>

</html>

## **Three Ways to Insert CSS**

There are three ways of inserting a style sheet:

* External CSS
* Internal CSS
* Inline CSS

## **External CSS**

With an external style sheet, you can change the look of an entire website by changing just one file!

Each HTML page must include a reference to the external style sheet file inside the **<link>** element, inside the head section.

Create a folder, css and save mystyle.css in this folder

mystyle.css

—-----------------------------------

body {

background-color: lightblue;

}

h1 {

color: navy;

margin-left: 20px;

}

Create a folder, html and place the following file in this folder.

—-----------------------------demo.html—-------------------------

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" href="../css/mystyle.css">

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

—------------------------------------------------------------------------------------------------------------

## **Internal CSS**

An internal style sheet may be used if one single HTML page has a unique style.

The internal style is defined inside the <style> element, inside the head section.

<!DOCTYPE html>

<html>

<head>

**<style>**

**body {**

**background-color: linen;**

**}**

**h1 {**

**color: maroon;**

**margin-left: 40px;**

**}**

**</style>**

</head>

<body>

<h1>This is a heading</h1>

<p>This is a paragraph.</p>

</body>

</html>

## **Inline CSS**

An inline style may be used to apply a unique style for a single element.

To use inline styles, add the style attribute to the relevant element. The style attribute can contain any CSS property.

<!DOCTYPE html>

<html>

<body>

<h1 style="color:blue;text-align:center;">This is a heading</h1>

<p style="color:red;">This is a paragraph.</p>

<p>This is styling **<span style="color:green;">applied to part of a </span>**text</p>

</body>

</html>

**—---------------------------------------------------------------------------------------------------**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<link rel="stylesheet" type="text/css" href="../css/mystyle.css">**

**<style>**

**h1 {**

**color: orange;**

**}**

**</style>**

**</head>**

**<body>**

**<h1>This is a heading</h1>**

**<p>The style of this document is a combination of an external stylesheet, and internal style</p>**

**</body>**

**</html>**

## **Cascading Order**

What style will be used when there is more than one style specified for an HTML element?

All the styles in a page will "cascade" into a new "virtual" style sheet by the following rules, where number one has the highest priority:

1. Inline style (inside an HTML element)
2. External and internal style sheets (in the head section)
3. Browser default

So, an inline style has the highest priority, and will override external and internal styles and browser defaults.

## **CSS Background Color**

<h1 style="background-color:DodgerBlue;">Hello World</h1>

<p style="background-color:Tomato;">Lorem ipsum...</p>

## **CSS Text Color**

<h1 style="color:Tomato;">Hello World</h1>

<p style="color:DodgerBlue;">Lorem ipsum...</p>

<p style="color:MediumSeaGreen;">Ut wisi enim...</p>

## **CSS Border Color**

<h1 style="border:2px solid Tomato;">Hello World</h1>

<h1 style="border:2px solid DodgerBlue;">Hello World</h1>

<h1 style="border:2px solid Violet;">Hello World</h1>

## **CSS Color Values**

In CSS, colors can also be specified using RGB values, HEX values, HSL values, RGBA values, and HSLA values:

<h1 style="background-color:rgb(255, 99, 71);">...</h1>

<h1 style="background-color:#ff6347;">...</h1>

<h1 style="background-color:hsl(9, 100%, 64%);">...</h1>

<h1 style="background-color:rgba(255, 99, 71, 0.5);">...</h1>

<h1 style="background-color:hsla(9, 100%, 64%, 0.5);">...</h1>

CSS Tables

<!DOCTYPE html>

<html>

<head>

**<style>**

**table, td, th {**

**border: 1px solid;**

**}**

**table {**

**width: 100%;**

**border-collapse: collapse;**

**}**

**</style>**

</head>

<body>

<h2>Let the table borders collapse</h2>

<table>

<tr>

<th>Firstname</th>

<th>Lastname</th>

</tr>

<tr>

<td>Peter</td>

<td>Griffin</td>

</tr>

<tr>

<td>Lois</td>

<td>Griffin</td>

</tr>

</table>

</body>

</html>

—-----------------------------------------------------------------------------------

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**table, td, th {**

**border: 1px solid black;**

**}**

**table {**

**border-collapse: collapse;**

**width: 100%;**

**}**

**th {**

**height: 70px;**

**}**

**</style>**

**</head>**

**<body>**

**<h2>The width and height Properties</h2>**

**<p>Set the width of the table, and the height of the table header row:</p>**

**<table>**

**<tr>**

**<th>Firstname</th>**

**<th>Lastname</th>**

**<th>Savings</th>**

**</tr>**

**<tr>**

**<td>Peter</td>**

**<td>Griffin</td>**

**<td>$100</td>**

**</tr>**

**<tr>**

**<td>Lois</td>**

**<td>Griffin</td>**

**<td>$150</td>**

**</tr>**

**<tr>**

**<td>Joe</td>**

**<td>Swanson</td>**

**<td>$300</td>**

**</tr>**

**<tr>**

**<td>Cleveland</td>**

**<td>Brown</td>**

**<td>$250</td>**

**</tr>**

**</table>**

**</body>**

**</html>**

## **Hoverable Table**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**table {**

**border-collapse: collapse;**

**width: 100%;**

**}**

**th, td {**

**padding: 8px;**

**text-align: left;**

**border-bottom: 1px solid #ddd;**

**}**

**tr:hover {background-color: coral;}**

**</style>**

**</head>**

**<body>**

**<h2>Hoverable Table</h2>**

**<p>Move the mouse over the table rows to see the effect.</p>**

**<table>**

**<tr>**

**<th>First Name</th>**

**<th>Last Name</th>**

**<th>Points</th>**

**</tr>**

**<tr>**

**<td>Peter</td>**

**<td>Griffin</td>**

**<td>$100</td>**

**</tr>**

**<tr>**

**<td>Lois</td>**

**<td>Griffin</td>**

**<td>$150</td>**

**</tr>**

**<tr>**

**<td>Joe</td>**

**<td>Swanson</td>**

**<td>$300</td>**

**</tr>**

**<tr>**

**<td>Cleveland</td>**

**<td>Brown</td>**

**<td>$250</td>**

**</tr>**

**</table>**

**</body>**

**</html>**

**—------------------------------------------------------------------------------------------------------------**

# **CSS Pseudo-classes**

## **What are Pseudo-classes?**

**A pseudo-class is used to define a special state of an element.**

**For example, it can be used to:**

* **Style an element when a user mouses over it**
* **Style visited and unvisited links differently**
* **Style an element when it gets focus**

## **Syntax**

**The syntax of pseudo-classes:**

**selector:pseudo-class {**

**property: value;**

**}**

## **Anchor Pseudo-classes**

**Links can be displayed in different ways:**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**/\* unvisited link \*/**

**a:link {**

**color: red;**

**}**

**/\* visited link \*/**

**a:visited {**

**color: green;**

**}**

**/\* mouse over link \*/**

**a:hover {**

**color: hotpink;**

**}**

**/\* selected link \*/**

**a:active {**

**color: blue;**

**}**

**</style>**

**</head>**

**<body>**

**<h2>Styling a link depending on state</h2>**

**<p><b><a href="default.asp" target="\_blank">This is a link</a></b></p>**

**<p><b>Note:</b> a:hover MUST come after a:link and a:visited in the CSS definition in order to be effective.</p>**

**<p><b>Note:</b> a:active MUST come after a:hover in the CSS definition in order to be effective.</p>**

**</body>**

**</html>**

# **CSS Navigation Bar**

## **Navigation Bar = List of Links**

# **CSS Vertical Navigation Bar**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**body {**

**margin: 0;**

**}**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**width: 25%;**

**background-color: #f1f1f1;**

**position: fixed;**

**height: 100%;**

**overflow: auto;**

**}**

**li a {**

**display: block;**

**color: #000;**

**padding: 8px 16px;**

**text-decoration: none;**

**}**

**li a.active {**

**background-color: #04AA6D;**

**color: white;**

**}**

**li a:hover:not(.active) {**

**background-color: #555;**

**color: white;**

**}**

**</style>**

**</head>**

**<body>**

**<ul>**

**<li><a class="active" href="#home">Home</a></li>**

**<li><a href="#news">News</a></li>**

**<li><a href="#contact">Contact</a></li>**

**<li><a href="#about">About</a></li>**

**</ul>**

**<div style="margin-left:25%;padding:1px 16px;height:1000px;">**

**<h2>Fixed Full-height Side Nav</h2>**

**<h3>Try to scroll this area, and see how the sidenav sticks to the page</h3>**

**<p>Notice that this div element has a left margin of 25%. This is because the side navigation is set to 25% width. If you remove the margin, the sidenav will overlay/sit on top of this div.</p>**

**<p>Also notice that we have set overflow:auto to sidenav. This will add a scrollbar when the sidenav is too long (for example if it has over 50 links inside of it).</p>**

**<p>Some text..</p>**

**<p>Some text..</p>**

**<p>Some text..</p>**

**<p>Some text..</p>**

**<p>Some text..</p>**

**<p>Some text..</p>**

**<p>Some text..</p>**

**</div>**

**</body>**

**</html>**

**CSS horizontal Nav Bar**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**ul {**

**list-style-type: none;**

**margin: 0;**

**padding: 0;**

**overflow: hidden;**

**border: 1px solid #e7e7e7;**

**background-color: #f3f3f3;**

**}**

**li {**

**float: left;**

**}**

**li a {**

**display: block;**

**color: #666;**

**text-align: center;**

**padding: 14px 16px;**

**text-decoration: none;**

**}**

**li a:hover:not(.active) {**

**background-color: #ddd;**

**}**

**li a.active {**

**color: white;**

**background-color: #04AA6D;**

**}**

**</style>**

**</head>**

**<body>**

**<ul>**

**<li><a class="active" href="#home">Home</a></li>**

**<li><a href="#news">News</a></li>**

**<li><a href="#contact">Contact</a></li>**

**<li><a href="#about">About</a></li>**

**</ul>**

**</body>**

**</html>**

# **CSS Forms**

**<!DOCTYPE html>**

**<html>**

**<style>**

**input[type=text], select {**

**width: 100%;**

**padding: 12px 20px;**

**margin: 8px 0;**

**display: inline-block;**

**border: 1px solid #ccc;**

**border-radius: 4px;**

**box-sizing: border-box;**

**}**

**input[type=submit] {**

**width: 100%;**

**background-color: #4CAF50;**

**color: white;**

**padding: 14px 20px;**

**margin: 8px 0;**

**border: none;**

**border-radius: 4px;**

**cursor: pointer;**

**}**

**input[type=submit]:hover {**

**background-color: #45a049;**

**}**

**div {**

**border-radius: 5px;**

**background-color: #f2f2f2;**

**padding: 20px;**

**}**

**</style>**

**<body>**

**<h3>Using CSS to style an HTML Form</h3>**

**<div>**

**<form action="/action\_page.php">**

**<label for="fname">First Name</label>**

**<input type="text" id="fname" name="firstname" placeholder="Your name..">**

**<label for="lname">Last Name</label>**

**<input type="text" id="lname" name="lastname" placeholder="Your last name..">**

**<label for="country">Country</label>**

**<select id="country" name="country">**

**<option value="australia">Australia</option>**

**<option value="canada">Canada</option>**

**<option value="usa">USA</option>**

**</select>**

**<input type="submit" value="Submit">**

**</form>**

**</div>**

**</body>**

**</html>**

**—------------------------------------------------------------------------------------------------------------**

**Search field example**

**<!DOCTYPE html>**

**<html>**

**<head>**

**<style>**

**input[type=text] {**

**width: 100%;**

**box-sizing: border-box;**

**border: 2px solid #ccc;**

**border-radius: 4px;**

**font-size: 16px;**

**background-color: white;**

**background-image: url('searchicon.png');**

**background-position: 10px 10px;**

**background-repeat: no-repeat;**

**padding: 12px 20px 12px 40px;**

**}**

**</style>**

**</head>**

**<body>**

**<h2>Input field with an icon inside</h2>**

**<form>**

**<input type="text" name="search" placeholder="Search..">**

**</form>**

**</body>**

**</html>**

**—------------------------------------------------------------------------------------------------------------ Bootstrap 5**

**Bootstrap 5 is the newest version of** [**Bootstrap**](https://www.w3schools.com/bootstrap/default.asp)**, which is the most popular HTML, CSS, and JavaScript framework for creating responsive, mobile-first websites.**

**Other popular libraries:**

**Bulma, Tailwind, Material UI etc.**

[**https://www.w3schools.com/bootstrap5/tryit.asp?filename=trybs\_default&stacked=h**](https://www.w3schools.com/bootstrap5/tryit.asp?filename=trybs_default&stacked=h)

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<title>Bootstrap 5 Example</title>**

**<meta charset="utf-8">**

**<meta name="viewport" content="width=device-width, initial-scale=1">**

**<link href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/css/bootstrap.min.css" rel="stylesheet">**

**<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.3/dist/js/bootstrap.bundle.min.js"></script>**

**</head>**

**<body>**

**<div class="container-fluid p-5 bg-primary text-white text-center">**

**<h1>My First Bootstrap Page</h1>**

**<p>Resize this responsive page to see the effect!</p>**

**</div>**

**<div class="container mt-5">**

**<div class="row">**

**<div class="col-sm-4">**

**<h3>Column 1</h3>**

**<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>**

**<p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>**

**</div>**

**<div class="col-sm-4">**

**<h3>Column 2</h3>**

**<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>**

**<p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>**

**</div>**

**<div class="col-sm-4">**

**<h3>Column 3</h3>**

**<p>Lorem ipsum dolor sit amet, consectetur adipisicing elit...</p>**

**<p>Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris...</p>**

**</div>**

**</div>**

**</div>**

**</body>**

**</html>**

**Topics to go through:**

**Bootstrap**

**Containers, Tables, Images, Buttons, Pagination,**

**Cards, NavBar, Carousel, Forms, Form Validation**

**Exercise:**

Design HTML form with styling for registering persons into a database.

Include the fields of Person class(Java) plus additional fields.

The form should contain the following fields:

AdharCardNumber, firstName, lastName, gender, birthdate, address, religion, city/town/village, state, pincode, education, email, mobile, password

—-------------------------register.css—---------------------------------------------------------

body{

background-color: #dee9ff;

}

.registration-form{

padding: 50px 0;

}

.registration-form form{

background-color: #fff;

max-width: 600px;

margin: auto;

padding: 50px 70px;

border-top-left-radius: 30px;

border-top-right-radius: 30px;

box-shadow: 0px 2px 10px rgba(0, 0, 0, 0.075);

}

.registration-form .form-icon{

text-align: center;

background-color: #5891ff;

border-radius: 50%;

font-size: 40px;

color: white;

width: 100px;

height: 100px;

margin: auto;

margin-bottom: 50px;

line-height: 100px;

}

.registration-form .item{

border-radius: 20px;

margin-bottom: 25px;

padding: 10px 20px;

}

.registration-form .create-account{

border-radius: 30px;

padding: 10px 20px;

font-size: 18px;

font-weight: bold;

background-color: #5791ff;

border: none;

color: white;

margin-top: 20px;

}

.registration-form .social-media{

max-width: 600px;

background-color: #fff;

margin: auto;

padding: 35px 0;

text-align: center;

border-bottom-left-radius: 30px;

border-bottom-right-radius: 30px;

color: #9fadca;

border-top: 1px solid #dee9ff;

box-shadow: 0px 2px 10px rgba(0, 0, 0, 0.075);

}

.registration-form .social-icons{

margin-top: 30px;

margin-bottom: 16px;

}

.registration-form .social-icons a{

font-size: 23px;

margin: 0 3px;

color: #5691ff;

border: 1px solid;

border-radius: 50%;

width: 45px;

display: inline-block;

height: 45px;

text-align: center;

background-color: #fff;

line-height: 45px;

}

.registration-form .social-icons a:hover{

text-decoration: none;

opacity: 0.6;

}

@media (max-width: 576px) {

.registration-form form{

padding: 50px 20px;

}

.registration-form .form-icon{

width: 70px;

height: 70px;

font-size: 30px;

line-height: 70px;

}

}

—--------------------------------------------------registration.html—------------

<!DOCTYPE html>

<html>

<head>

<meta charset="utf-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1">

<title>The Easiest Way to Add Input Masks to Your Forms</title>

<link href="https://cdnjs.cloudflare.com/ajax/libs/simple-line-icons/2.4.1/css/simple-line-icons.min.css" rel="stylesheet">

<link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0/css/bootstrap.min.css">

<link rel="stylesheet" href="../css/register.css">

</head>

<body>

<div class="registration-form">

<form id="frmid" action="#" method="GET">

<div class="form-group" align="center">

<h4>Registration Form</h4>

</div>

<div class="form-group">

<input type="text" class="form-control item" id="username" placeholder="Username">

</div>

<div class="form-group">

<input type="password" class="form-control item" id="password" placeholder="Password">

</div>

<div class="form-group">

<input type="email" class="form-control item" id="email" placeholder="Email">

</div>

<div class="form-group">

<input type="text" class="form-control item" id="phonenumber" placeholder="Phone Number">

</div>

<div class="form-group">

<input type="date" class="form-control item" id="birthdate" placeholder="Birth Date">

</div>

<div class="form-group">

<button type="button" class="btn btn-block create-account">Register</button>

</div>

</form>

<!-- <div class="social-media">

<h5>Sign up with social media</h5>

<div class="social-icons">

<a href="#"><i class="icon-social-facebook" title="Facebook"></i></a>

<a href="#"><i class="icon-social-google" title="Google"></i></a>

<a href="#"><i class="icon-social-twitter" title="Twitter"></i></a>

</div>

</div> -->

</div>

<!-- <script type="text/javascript" src="https://code.jquery.com/jquery-3.2.1.min.js"></script> -->

<script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/jquery.mask/1.14.15/jquery.mask.min.js"></script>

</body>

</html>

<https://getbootstrap.com/docs/5.0/getting-started/introduction/>

[**https://bootsnipp.com**](https://bootsnipp.com)

[**https://epicbootstrap.com/snippets/registration**](https://epicbootstrap.com/snippets/registration)

**—----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------- Javascript**

[**https://www.javascripttutorial.net/what-is-javascript/**](https://www.javascripttutorial.net/what-is-javascript/)

[**https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/JavaScript\_Introduction.html**](https://www3.ntu.edu.sg/home/ehchua/programming/webprogramming/JavaScript_Introduction.html)

[**https://www.codewithharry.com/blogpost/javascript-cheatsheet/**](https://www.codewithharry.com/blogpost/javascript-cheatsheet/)

1. **Interpreter:** An interpreter runs instructions directly from the programming language without changing them into machine code first.

2. **Compiler:** A compiler changes the entire program into object code (or binary code) and saves it. This code can then be run by the machine. Static compilation.

3. **JIT Compiler**: A JIT compiler converts code into byte code first. Then, at runtime, it changes the byte code into machine-readable code, which makes the program run faster. Dynamic Compilation.

JavaScript is mainly interpreted, but modern JavaScript engines, like V8 in Google Chrome, use JIT (Just-In-Time) compilation to boost performance.

They convert JavaScript code into optimised machine code right before it runs. This mix of interpretation and JIT compilation makes JavaScript fast and versatile for web applications.

Hence we can conclude that JS code is executed in three phases.

**Parsing -> Compiling -> Executing**

**JavaScript code is initially interpreted before any execution begins.**

Note: To enable paste feature in Firefox browser console, first type “allow paste”

{ console.log("Hello"); console.log("Hello Again"); var hello="HELLO"; console.log(hello); console.log("Thank You") }

Notable JavaScript engines include V8 in Chrome, SpiderMonkey in Firefox, and JavaScriptCore in Safari.

## **Client-side vs. Server-side JavaScript**

When JavaScript is used on a web page, it is executed in web browsers, serving as a client-side language.

JavaScript can run on both web browsers and servers. A popular JavaScript server-side environment is [Node.js](https://www.javascripttutorial.net/nodejs-tutorial/). Unlike client-side JavaScript, server-side JavaScript executes on the server and allows you to access databases, file systems, etc.

JavaScript is the most widely used *client-side* programming language that lets you supercharge your HTML with interactivity, animation and dynamic visual effect for better User Interface and User Experience (UI/UX). It is:

* a small, lightweight, object-oriented, cross-platform, special-purpose scripting language meant to be run under a host environment (typically a web browser).
* a *client-side scripting language* to enrich web user-interfaces and create dynamic web pages (e.g., for...input validation, and immediate response to user's actions).
* the engine that supports AJAX (Asynchronous JavaScript and XML - that can be used to update one part of the web page asynchronously), which generate renew interest in JavaScript.
* **JavaScript is Now Everywhere with Node.js**

JavaScript has grown beyond the client-side programming. With the introduction of Node.js in 2009 (an open-source, cross-platform JavaScript run-time environment), you can run your JavaScript standalone or inside the server (instead of a browser). This allows you to use one single language for both the server-side and client-side programming.

**Built-in objects provided by underlying OS.**

window

document

console

—------------------------------------------------------------------------------

### **JavaScript Basic Syntax**

#### **Comments**

Comments are ignored by the JavaScript runtime but greatly useful in explaining your codes to others (and also to yourself three days later). You should use comments liberally to explain or document your codes.

An *end-of-line comment* begins with // and lasts till the end of the current line. A *multi-line comment* begins with /\* and lasts till \*/.

Take note that:

* HTML comments are enclosed inside <!-- ... -->.
* CSS supports multi-line comment /\* ... \*/, but NOT end-of-line comment //.
* JavaScript supports both /\* ... \*/ and //, like Java/C/C++/C#.

#### **Whitespaces (blank, tab, newline)**

Like C/C++/Java, JavaScript ignores additional whitespaces (blanks, tabs, and newlines). I strongly recommend that you use additional whitespaces to format your program to make your code easier to read and understand.

#### **Expressions**

An *expression* is a combination of *variables*, *literals*, *operators*, and *sub-expressions* that can be evaluated to produce *a single value*.

#### **Statements, Semicolon and Blocks**

A *statement* is a single programming instruction. Unlike C/C++/Java, where you need to end a statement with a semicolon (;), in JavaScript the semicolon is *optional*. However, if semicolon is missing, you need to end the statement with a newline (and JavaScript engine will insert a semicolon for you).

Semicolons in JavaScript divide the community. Some prefer to use them always, no matter what. Others like to avoid them for brevity. I shall leave it to you.

A *block* consists of zero or more statements enclosed in a pair of curly braces { *statements* }. No semicolon is needed after the closing brace.

#### **Variables, Literals & Types**

##### **Variable declarations (var, let, const) and Assignment Operator (=)**

A variable is a *named* storage location that holds a *value*. Prior to ES6, you can only use var to declare global variables. ES6 introduces two new keywords: let to declare a *local block-scope* variable, and const to declare a local block-scope constant.

You should try const (safest), followed by let, and avoid var.

You can assign (and re-assign) a value to a variable using the assignment (=) operator. For example,

const magic = 88; // const-variable's value cannot change

for (let i = 0; i < 5; ++i) { // i has local block-scope

console.log(i);

}

var font\_size = 12; // var-variable is global

##### **Identifiers**

Identifiers are names given to identify entities (such as variables and functions). The rules for valid identifiers are:

* An identifier can contain letters (a-z, A-Z), digits (0-9), underscore (\_) and dollar sign ($). But it cannot begin with a digit (0-9).
* Identifiers are case-sensitive. A rose is NOT a Rose, and is NOT a ROSE.
* Identifiers cannot be keywords.
* Take note that hyphen (-) and space are NOT allowed. Hence, font-size, roman new, are NOT valid identifiers. You should use underscore (\_) instead of hyphen (-).

##### **Literals**

A *literal* is a fixed value, e.g., 5566, 3.14, "Hello", 'apple', true, null, that can be assigned to a variable, or form part of an expression.

##### **Types**

JavaScript is object-oriented. But, It supports both *primitive types* and *objects*.

Primitives are not objects and do not possess properties and methods. JavaScript supports these primitive types:

1. string: a sequence of characters. Strings literals are enclosed in a pair of single quotes or double quotes (e.g., "Hello", 'world').
2. number: takes both integer (e.g., 5566) or floating-point (e.g., 3.14159265).
3. boolean: takes boolean literal of either true or false (in lowercase).
4. undefined: takes a special literal value called undefined. Take note that undefined is both a type and a literal value.
5. symbol (ES6): A data type whose instances are unique and immutable.
6. bigint (ES2020/ES11): integers with arbitrary precision.

JavaScript also supports these object types and value (we shall discuss object later):

1. object: for general objects.
2. function: for function objects. Unlike Java, function is a *first class object* in JavaScript, e.g., you can assign a function to a variable.
3. null: A special literal value for unallocated (unconstructed) object. Take note that null is NOT undefined. The typeof(null) is object. null is meant to represent an absence of a constructed object. null is actually considered to be a primitive as it has no properties and methods.

Unlike most of the general programming languages (such as Java/C/C++/C#) which are strongly type, JavaScript is *loosely type* (similar to most of the scripting languages such as UNIX Shell Script, Perl, Python). You do not have to explicitly declare the type of a variable (such as int and float) during declaration. The type is decided *when a value is assigned to that variable*. If a number is assigned, the variable takes on the number type and can perform numeric operations such as addition and subtraction. If a string is assigned, the variable takes on the string type and can perform string operations such as string concatenation. In other words, *the type is associated with the value, instead of the variable*.

##### **Operator typeof**

You can use the operator typeof to check the type of the current value assigned to a variable. typeof returns a string.

typeof *varname* // typeof is an operator

typeof(*varname*) // You can also invoke using function format

For example,

let msg = 'hello'

console.log(typeof msg) //string

let num = 5

console.log(typeof (num)) //number

—-----------------------------------------------------------------------------------------

**!DOCTYPE html>**

**<html>**

**<head>**

**<meta charset="utf-8">**

**<script>**

**alert("Welcome To Javascript");**

**</script>**

**</head>**

**<body>**

**<script>**

**document.write("I'm coming from Javascript code-Line1");**

**</script>**

**<h2>My Home Page</h2>**

**<script>**

**document.write("Last Modifed on "+ document.lastModified);**

**</script>**

**</body>**

**</html>**

**—--------------------------------------------------------------------------------------------------------------**

There are three kinds of pop-up *dialog* boxes for interacting with the users:

1. The alert(*str*) function puts the *str* on a pop-up box with a OK button. User needs to click the OK button to continue.
2. The prompt(*promptingStr*, *defaultStr*?) function puts up an input pop-up box with the *promptingStr* with an OK and Cancel buttons. It returns the input entered by the user as a string; or a special value called null if the user hits the Cancel button. The optional parameter *defaultStr* specifies the initial string to be shown.
3. The confirm(*str*) function puts *str* on a pop-up box with OK and Cancel buttons. It returns true if user hits the OK button; or false otherwise.

**<!DOCTYPE html>**

**<!-- JSExVar.html -->**

**<html lang="en">**

**<head>**

**<meta charset="utf-8">**

**<title>JavaScript Example: Variables and functions prompt() and confirm()</title>**

**<script>**

**var username = prompt("Enter your name: ", "");**

**if (confirm("Your name is " + username)) {**

**document.write("<h1>Hello, " + username + "!</h1>");**

**} else {**

**document.write("<h1>Hello, world!</h1>");**

**}**

**</script>**

**</head>**

**<body>**

**<p>Welcome to JavaScript!</p>**

**</body>**

**</html>**

—----------------------------------------------------------------------------------------------

**<!DOCTYPE html>**

**<!-- Date is a built-in Javascript object-->**

**<html lang="en">**

**<head>**

**<meta charset="utf-8">**

**<title>JavaScript Example: The Date object</title>**

**<script>**

**var now = new Date(); // current date/time**

**var hrs = now.getHours(); // 0 to 23**

**var mins = now.getMinutes();**

**var secs = now.getSeconds();**

**document.writeln("<p>It is " + now + "</p>");**

**document.writeln("<p>Hour is " + hrs + "</p>");**

**document.writeln("<p>Minute is " + mins + "</p>");**

**document.writeln("<p>Second is " + secs + "</p>");**

**if (hrs < 12) {**

**document.writeln("<h2>Good Morning!</h2>");**

**} else {**

**document.writeln("<h2>Good Afternoon!</h2>");**

**}**

**</script>**

**</head>**

**<body></body>**

**</html>**

# **Standard built-in Javascript objects**

### [**Value properties**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects#value_properties)

These global properties return a simple value. They have no properties or methods.

* [Infinity](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Infinity)
* [NaN](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/NaN)
* [undefined](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/undefined)

### [**Function properties**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects#function_properties)

These global functions—functions which are called globally, rather than on an object—directly return their results to the caller.

* [eval()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/eval)
* [isFinite()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/isFinite)
* [isNaN()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/isNaN)
* [parseFloat()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseFloat)
* [parseInt()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/parseInt)

### [**Fundamental objects**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects#fundamental_objects)

These objects represent fundamental language constructs.

* [Object](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Object)
* [Function](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Function)
* [Boolean](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Boolean)

### [**Numbers and dates**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects#numbers_and_dates)

These are the base objects representing numbers, dates, and mathematical calculations.

* [Number](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number)
* [BigInt](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/BigInt)
* [Math](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math)
* [Date](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date)

### [**Text processing**](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects#text_processing)

These objects represent strings and support manipulating them.

* [String](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String)
* [RegExp](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/RegExp)

**Use a conditional statement to print the day of the week in word (i.e., 0 for Sunday, 1 for Monday and etc.)**

**<!DOCTYPE html>**

**<!-- Date is a built-in Javascript object-->**

**<html lang="en">**

**<head>**

**<meta charset="utf-8">**

**<title>JavaScript Example: The Date object</title>**

**</head>**

**<body>**

**<p> Today is: </p>**

**<script>**

**var day = new Date().getDay(); // current date/time**

**switch(day){**

**case 0: document.writeln("Sunday"); break;**

**case 1: document.writeln("Monday"); break;**

**case 2: document.writeln("Tuesday"); break;**

**case 3: document.writeln("Wednesday"); break;**

**case 4: document.writeln("Thursday"); break;**

**case 5: document.writeln("Friday"); break;**

**case 6: document.writeln("Saturday"); break;**

**}**

**</script>**

**</body>**

**</html>**

—--------------------------==============================================

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>JavaScript Example: Loop</title>

</head>

<body>

<h2>Testing Loop</h2>

<script>

var multiplier = prompt("Enter a multiplier: ");

for (var number = 1; number <= 100; number++) {

document.writeln(number \* multiplier);

}

</script>

</body>

</html>

==============================================================================

Declaring variables using **var** and **let** keywords

var is function-scoped while let is block-scoped.

Declaring a variable inside a function with **var** keyword will limit the scope of the variable to the function i.e cannot be accessed outside the function.

Note that variable declared with var keyword outside a function is called a global variable and is stored in **window** object

A block is one or more statements placed within curly braces {}.

Variable declared inside a block with **let** keyword is accessible only within that block.

**var x=5;**

**{**

**/\* defining function \*/**

**function sampleFn(){**

**var x=10;**

**console.log(x);**

**}**

**sampleFn(); //calling the function**

**console.log(window.x);**

**}**

—--------------------------------------------------------------------------------------------

var x=5;

{

/\* defining function \*/

function sampleFn(){

var x=10;

var y=x+5;

console.log(x);

}

sampleFn(); //calling the function

console.log(window.x); // refers to global variable, same as console.log(x);

**console.log(y); //error**

}

—----------------------------------------------------------------------------------------------------

let x111=5;

{

/\* defining function \*/

function sampleFn(){

let x=10;

{

let y=x+5;

}

// console.log(x+","+y); y is not defined

}

sampleFn(); //calling the function

//global variable declared with let keyword is not placed in window object

console.log(x111);

}

—----------------------------------------------------------------------------------------------------------------------------------

**const keyword**

**Like let const is also block-scoped variable**

**Declaring a constant:**

**const constant\_name = value;**

Note: C, C++, Java, C# are block-scoped languages i.e variables and constants scope is limited to the block where it is declared.

{

// Declare variables with initial values

let f\_name = "Alex";

const ZIP = 560089;

var age = 25;

// Reassign values

f\_name = "Bob"; // the f\_name value is 'Bob"

// ZIP = 65457; // Uncaught TypeError: Assignment to constant variable.

age = 78; // the age value is 78

console.log(f\_name+","+ZIP+","+age);

}

—------------------------------------------------------------------

**{**

**console.log(x); // output as undefined**

**var x=5;**

**console.log(y); //error,can't access lexical declaration 'y' before initialization**

**let y=10;**

**}**

**—----------------------------------------------------------------------**

**Note: The syntax of conditional controls ( if, if .. else, nested if, if else if else ladder, switch statement) and iterative controls( while, do ..while and for loop) are same as in C, C++ and Java.**

**—----------------------------------------------------------------------------------------**

**Javascript Arrays:**

**let arrayName = [<listOfValues>];**

**Ex.**

**Modifiable array:**

**let list = [1,2,3,4,5,6,7,8,9,10];**

**Unmodifiable array:**

**const list = [1,2,3,4,5,6,7,8,9,10];**

**Empty array:**

**list myList = [];**

**Creating an array with constructor:**

**const cars = new Array("Saab", "Volvo", "BMW");**

**same as**

**const cars = [ "Saab", "Volvo", "BMW"];**

**Note: Array indexing starts from 0.**

**Javascript methods on arrays**

## **Converting an Array to a String**

The JavaScript method toString() converts an array to a string of (comma separated) array values.

{

const cars = [ "Saab", "Volvo", "BMW"];

console.log(cars.toString());

}

**{**

**const fruits = ["Banana", "Orange", "Apple", "Mango"];**

**console.log(typeof(fruits)); //object**

**console.log(fruits.length);**

**console.log(fruits[0]);**

**console.log(fruits.at(0));**

**console.log(fruits.toString());**

**console.log(fruits.join("-"));**

**}**

**Popping items out of an array, or pushing items into an array**

**{**

**const fruits = ["Banana", "Orange", "Apple", "Mango"];**

**console.log(fruits);**

**let f1 = fruits.pop();**

**console.log(f1);**

**console.log(fruits);**

**fruits.push("Kiwi");**

**console.log(fruits);**

**}**

**shift() and unshift() methods**

shift() removes the first element and left-shifts the remaining elements.

unshift() adds new element in the first position and shifts all the elements to right by one position

{

const fruits = ["Banana", "Orange", "Apple", "Mango"];

console.log(fruits);

let f1 = fruits.shift();

console.log(f1);

console.log(fruits);

fruits.unshift("Kiwi");

console.log(fruits);

}

**delete() method : Use above methods rather than delete() method since it leaves holes within the array.**

**{**

**const fruits = ["Banana", "Orange", "Apple", "Mango"];**

**console.log(fruits);**

**let f1 = fruits.shift();**

**console.log(f1);**

**console.log(fruits);**

**fruits.unshift("Kiwi");**

**console.log(fruits);**

**delete fruits[0];**

**console.log(fruits.length); //4**

**console.log(fruits);//<1 empty slot>, "Orange", "Apple", "Mango" ]**

**}**

## **JavaScript Array concat()**

**The concat() method creates a new array by merging (concatenating) existing arrays:**

### **{**

**const myGirls = ["Cecilie", "Lone"];**

**const myBoys = ["Emil", "Tobias", "Linus"];**

**const myChildren = myGirls.concat(myBoys);**

**console.log(myChildren);**

**}**

## **JavaScript Array flat()**

**The flat() method creates a new array with sub-array elements concatenated to a specified depth.**

**{**

**const TwoDfruits = [["Banana", "Orange"], ["Apple", "Mango"]];**

**console.log(TwoDfruits);**

**const linearFruits = TwoDfruits.flat();**

**console.log(linearFruits);**

**}**

## **Splicing and Slicing Arrays**

**The splice() method adds new items to an array.**

**The slice() method slices out a piece of an array.**

## **JavaScript Array splice()**

**The splice() method can be used to add new items to an array:**

**{**

**const TwoDfruits = [["Banana", "Orange"], ["Apple", "Mango"]];**

**console.log(TwoDfruits);**

**const linearFruits = TwoDfruits.flat();**

**console.log(linearFruits);**

**linearFruits.splice(2,0,"Kiwi","Grapes");**

**console.log(linearFruits); //[ "Banana", "Orange", "Kiwi", "Grapes", "Apple", "Mango" ]**

**}**

**{**

**const fruits = ["Banana", "Orange", "Apple", "Mango"];**

**let f2 =fruits.splice(2, 2, "Lemon", "Kiwi");**

**console.log(fruits); console.log("Removed elements: " + f2);**

**}**

**Array(4) [ "Banana", "Orange", "Lemon", "Kiwi" ]**

**{**

**const fruits = ["Banana", "Orange", "Lemon", "Apple", "Mango"];**

**const citrus = fruits.slice(1);**

**console.log(fruits);**

**console.log(citrus);**

**}**

**Array(5) [ "Banana", "Orange", "Lemon", "Apple", "Mango" ]**

**debugger eval code:5:9**

**Array(4) [ "Orange", "Lemon", "Apple", "Mango" ]**

## **Array Find and Search Methods**

| [**indexOf()**](https://www.w3schools.com/js/js_array_search.asp#mark_indexof)**, find()**  [**lastIndexOf()**](https://www.w3schools.com/js/js_array_search.asp#mark_lastindexof)  [**includes()**](https://www.w3schools.com/js/js_array_search.asp#mark_includes)  **{**  **const numbers = [4, 9, 16, 25, 29];**  **let first = numbers.find(myFunction);**  **function myFunction(value, index, array) {**  **return value > 18;**  **}**  **console.log(first);**  **}** **Sorting an Array** **The sort() method sorts an array alphabetically:**  **New Method: toSorted()**  **Reversing an Array, reverse()** | [**find()**](https://www.w3schools.com/js/js_array_search.asp#mark_find)  [**findIndex()**](https://www.w3schools.com/js/js_array_search.asp#mark_findindex)  [**findLast()**](https://www.w3schools.com/js/js_array_search.asp#mark_findlast)  [**findLastIndex()**](https://www.w3schools.com/js/js_array_search.asp#mark_findlastindex) |
| --- | --- |

**New method: tpReversed()**

**{**

**const fruits = ["Banana", "Orange", "Apple", "Mango"];**

**fruits.sort();**

**console.log(fruits);**

**fruits.reverse();**

**console.log(fruits);**

**let fruitsNew = fruits.toSorted();**

**console.log(fruits);**

**console.log(fruitsNew);**

**let reversedFruits = fruits.toreversed();**

**console.log(fruits);**

**console.log(reversedFruits);**

**}**

## **Numeric Sort**

**By default, the sort() function sorts values as strings.**

**We need to pass callback function**

**Note: A callback is a function that is passed to another function as an argument.**

**{**

**const points = [40, 100, 1, 5, 25, 10];**

**points.sort(function(a, b){return a - b});**

**console.log(points); // [ 1, 5, 10, 25, 40, 100 ]**

**}**

## **Array Iteration Methods**

Note: We need to pass callback functions to the following methods.

**forEach(), map(), filter(), reduce()**

**Spread operator( … )**

**{**

**const numbers = [1,2,3,4,5];**

**numbers.forEach(myFunction );**

**function myFunction(value, index, array) {**

**console.log(value);**

**}**

**}**

## **JavaScript Array map()**

**The map() method creates a new array by performing a function on each array element.**

**{**

**const numbers1 = [45, 4, 9, 16, 25];**

**const numbers2 = numbers1.map(myFunction);**

**function myFunction(value, index, array) {**

**return value \* 2;**

**}**

**console.log(numbers1);**

**console.log(numbers2);**

**}**

**{**

**const numbers1 = [45, 4, 9, 16, 25];**

**const numbers2 = numbers1.map(myFunction);**

**function myFunction(value, index, array) { return value \* 2; }**

**console.log(numbers1);**

**console.log(numbers2);**

**}**

**Array(5) [ 45, 4, 9, 16, 25 ]**

**debugger eval code:8:9**

**Array(5) [ 90, 8, 18, 32, 50 ]**

**{**

**const numbers1 = [1,2,3,4,5,6];**

**const numbers2= numbers1.flatMap(myFunction);**

**function myFunction(value) {**

**return value\*value;**

**}**

**console.log(numbers1);**

**console.log(numbers2);**

**}**

## **JavaScript Array filter()**

**The filter() method creates a new array with array elements that pass a test.**

**{**

**const numbers = [1,2,3,4,5,6,7,8,9,10];**

**const evenNumbers= numbers.filter(myFunction);**

**function myFunction(value,index, array) {**

**return value%2 == 0;**

**}**

**console.log(numbers);**

**console.log(evenNumbers); //[ 2, 4, 6, 8, 10 ]**

**}**

## **JavaScript Array reduce()**

**The reduce() method runs a function on each array element to produce (reduce it to) a single value.**

**The reduce() method works from left-to-right in the array.**

**{**

**const numbers = [45, 4, 9, 16, 25];**

**let sum = numbers.reduce(myFunction);**

**function myFunction(total, value) { return total + value; }**

**console.log(sum);**

**}**

**99**

## **JavaScript Array every()**

**The every() method checks if all array values pass a test.**

**{**

**const numbers = [45, 4, 9, 16, 25];**

**let allOver18 = numbers.every(myFunction);**

**function myFunction(value, index, array) { return value > 18; }**

**console.log(allOver18);**

**}**

**false**

**Note: Complementary function for the above is called some()**

## **JavaScript Array.from()**

**The Array.from() method returns an Array object from any object with a length property or any iterable object.**

**{**

**let name="ADMIN";**

**const arr = Array.from(name);**

**console.log(arr);**

**}**

**Array(5) [ "A", "D", "M", "I", "N" ]**

## **JavaScript Array Spread (...)**

**The ... operator expands an iterable (like an array) into more elements:**

**{**

**let list1=[1,2,3,4,5];**

**let list2=[6,7,8,9];**

**let list3 = [...list1 , ...list2];**

**console.log(list3); //[ 1, 2, 3, 4, 5, 6, 7, 8, 9 ]**

**}**

**—-------------------------------------------------------------------------------------------------------------------------**

**Javascript built-in objects**

**Math, Date, Random**

# **JavaScript Set and Map objects**

A JavaScript Set is a collection of unique values.

Each value can only occur once in a Set.

The values can be of any type, primitive values or objects.

{

const letters = new Set(["a","b","c"]);

console.log(letters);

letters.add("d");

letters.add("a");

console.log(letters);

const digits= new Set();

digits.add(1);

digits.add(2);

digits.add(1);

digits.add(2);

console.log(digits);

}

# **JavaScript Maps**

A Map holds key-value pairs where the keys can be any datatype.

A Map remembers the original insertion order of the keys.

{

// Create a Map

const fruits = new Map([

["apples", 500],

["bananas", 300],

["oranges", 200]

]);

console.log(fruits.get("bananas"));

// Create a Map

const fruitsNew = new Map();

// Set Map Values

fruitsNew.set("apples", 500);

fruitsNew.set("bananas", 300);

fruitsNew.set("oranges", 200);

console.log(fruits);

console.log(fruitsNew);

}

# **JavaScript Objects**

Objects are containers for Properties and Methods. Properties are named Values. Methods are Functions stored as Properties.

In JavaScript, almost "everything" is an object.

* Objects are objects
* Maths are objects
* Functions are objects
* Dates are objects
* Arrays are objects
* Maps are objects
* Sets are objects

**All JavaScript values, except primitives, are objects.**

## **JavaScript Primitives**

A **primitive value** is a value that has no properties or methods.

**3.14** is a primitive value

A **primitive data type** is data that has a primitive value.

JavaScript defines 7 types of primitive data types:

* string
* number
* boolean
* null
* undefined
* symbol
* bigint

| **Value** | **Type** | **Comment** |
| --- | --- | --- |
| "Hello" | string | "Hello" is always "Hello" |
| 3.14 | number | 3.14 is always 3.14 |
| true | boolean | true is always true |
| false | boolean | false is always false |
| null | null (object) | null is always null |
| undefined | undefined | undefined is always undefined |

**const objectName= {**

**name: value,**

**name: value,**

**……**

**};**

**Empty object**

**// Create an Object**

**const person = {};**

**// Add Properties**

**person.firstName = "John";**

**person.lastName = "Doe";**

**person.age = 50;**

**person.eyeColor = "blue";**

**{**

**const person = {**

**adharcard: 123456789,**

**name: "Smith",**

**birthdate: "10-15-2001",**

**gender: "M"**

**};**

**console.log(person.name+","+person.gender);**

**}**

**—---------------------------------------------------------------------------------------**

**{**

**//Javascript objects are mutable**

**const person = {**

**firstName:"John",**

**lastName:"Doe",**

**age:50, eyeColor:"blue"**

**}**

**//person and samePerson are referring to same object**

**const samePerson = person;**

**console.log(person.age+","+samePerson.age);**

**samePerson.age = 55;**

**console.log(person.age+","+samePerson.age);**

**}**

## **JavaScript Object Methods**

1. **Objects** are containers for **Properties** and **Methods**.
2. **Properties** are named **Values**.
3. **Methods** are **Functions** stored as **Properties**.
4. **Properties** can be primitive values, functions, or even other objects.

**const objectName = {**

**propertyName: value,**

**…….**

**functionName: function(){**

**}**

**……..**

**};**

**{**

**const person = {**

**firstName: "John",**

**lastName : "Doe",**

**id : 5566,**

**fullName : function() {**

**return this.firstName + " " + this.lastName;**

**}**

**};**

**console.log(person.fullName());**

**}**

**—----------------------------------------------------------------------------------------**

**{**

**const person = {**

**firstName: "John",**

**lastName : "Doe",**

**id : 5566,**

**fullName : function(firstName,lastName) {**

**this.firstName=firstName;**

**this.lastName=lastName;**

**return this.firstName + " " + this.lastName;**

**}**

**};**

**console.log(person.firstName+" "+person.lastName);**

**console.log(person.fullName("Ravi","Kumar"));**

**console.log(person.firstName+" "+person.lastName);**

**}**

**—----------------------------------------------------------------------------**

**{**

**const person = {**

**firstName: "John",**

**lastName : "Doe",**

**id : 5566,**

**fullName : function(firstName,lastName) {**

**this.firstName=firstName;**

**this.lastName=lastName;**

**return this.firstName + " " + this.lastName;**

**}**

**};**

**console.log(person.firstName+" "+person.lastName);**

**console.log(person.fullName("Ravi","Kumar"));**

**console.log(person.firstName+" "+person.lastName);**

**//adding property ao JS object**

**person.nationality="India";**

**//adding functionality to JS object**

**person.getId = function(){**

**return this.id;**

**}**

**console.log(person);**

**console.log(person.getId() );**

**//deleting property**

**delete person.lastName;**

**console.log(person);**

**}**

**{**

**const person = {**

**firstName: "John",**

**lastName : "Doe",**

**id : 5566,**

**fullName : function(firstName,lastName) {**

**this.firstName=firstName;**

**this.lastName=lastName;**

**//return this.firstName + " " + this.lastName;**

**return (this.firstName + " " + this.lastName).toUpperCase();**

**},**

**//Nested object**

**myCars: {**

**car1: "Ciaz",**

**car2: "Benz",**

**car3: "BMW"**

**}**

**};**

**console.log(person.myCars.car1+" "+person.myCars.car2+","+person.myCars.car3);**

**}**

**—----------------------------------------------------------------**

**Arrays of objects**

**const arrayName = [ {},{},{}];**

**Const persons = [**

**{**

**adharcard: 123456789,**

**name: "Smith",**

**birthdate: "10-15-2001",**

**gender: "M"**

**},**

**{**

**adharcard: 123456789,**

**name: "Smith",**

**birthdate: "10-15-2001",**

**gender: "M"**

**},**

**{**

**adharcard: 123456789,**

**name: "Smith",**

**birthdate: "10-15-2001",**

**gender: "M"**

**}**

**];**

Display the above objects

{

const person = {

adharcard: 123456789,

name: "Smith",

birthdate: "10-15-2001",

gender: "M"

};

console.log(person.name+","+person.gender);

const arr = Object.values(person);

console.log(arr);//[ 123456789, "Smith", "10-15-2001", "M" ]

}

{

// Create an Object

const person = {

name: "John",

age: 30,

city: "New York"

};

// Stringify Object

**let myString = JSON.stringify(person);**

console.log(person);

}

**Some solutions to display JavaScript objects are:**

* Displaying the Object Properties by name
* Displaying the Object Properties in a Loop
* Displaying the Object using Object.values()
* Displaying the Object using JSON.stringify()

**{**

**const person = {**

**name: "John",**

**age: 30,**

**city: "New York"**

**};**

**// Build a Text**

**let text = "";**

**for (let x in person) {**

**text += person[x] + " ";**

**};**

**console.log(text);**

**for(let i in person){**

**console.log(person[i]+",");**

**}**

**}**

—--------------------------------------------------------------

**Displaying the Object using Object.values()**

**Object.values() creates an array from the property values:**

{

const person = {

name: "John",

age: 30,

city: "New York"

};

//Object.values() creates an array from the property values:

const perArr = Object.values(person)

console.log(perArr);

}

## **Using Object.entries()**

Object.entries() makes it simple to use objects in loops:

**{**

**const fruits = {Bananas:300, Oranges:200, Apples:500};**

**let text = "";**

**for (let [name, value] of Object.entries(fruits)) {**

**text += name + ": " + value ;**

**}**

**console.log(text);**

**}**

## **Using JSON.stringify()**

**{**

**const fruits = {Bananas:300, Oranges:200, Apples:500};**

**console.log(JSON.stringify(fruits));**

**}**

# **JavaScript Object Constructors**

**{**

**//Object Type, Person**

**function Person(first, last, age, eye) {**

**this.firstName = first;**

**this.lastName = last;**

**this.age = age;**

**this.eyeColor = eye;**

**}**

**const myFather = new Person("John", "Doe", 50, "blue");**

**const myMother = new Person("Sally", "Rally", 48, "green");**

**const mySister = new Person("Anna", "Rally", 18, "green");**

**{"firstName":"John","lastName":"Doe","age":50,"eyeColor":"blue"}//**

**console.log(JSON.stringify(myFather));**

**}**

—-----------------------------------------------------------------------------------------------

**Javascript String object**

**let stringName = “literal”;**

**let stringName= ‘literal’;**

**Javascript String methods**

## **Extracting String Parts**

**There are 3 methods for extracting a part of a string:**

* **slice(*start*, *end*)**
* **substring(*start*, *end*)**
* **substr(*start*, *length*)**

## **JavaScript String search()**

**The search() method searches a string for a string (or a regular expression) and returns the position of the match:**

## **JavaScript String match()**

**{**

**let text = "The rain in SPAIN stays mainly in the plain";**

**const myArr = text.match(/ain/);**

**console.log(myArr.toString());**

**}**

**ain**

**—-------------------------------------------------------------------------------------------------------------------------------------------------------------------**

# **JavaScript Events**

**HTML events are "things" that happen to HTML elements.**

**When JavaScript is used in HTML pages, JavaScript can "react" to these events.**

## **HTML Events**

**An HTML event can be something the browser does, or something a user does.**

**Here are some examples of HTML events:**

* **An HTML web page has finished loading**
* **An HTML input field was changed**
* **An HTML button was clicked**

**When events happens, you may want to do something.**

**JavaScript lets you execute code when events are detected.**

**HTML allows event handler attributes, with JavaScript code, to be added to HTML elements.**

**<button onclick="document.getElementById('demo').innerHTML = Date()">The time is?</button>**

**JavaScript can be used to handle many types of events, in response to a user's action or browser's action. For example,**

* **onload: fires *after* browser loaded the page.**
* **onmouseover and onmouseout: fires when the user points the mouse pointer at/away from the HTML element.**

## **Common HTML Events**

**Here is a list of some common HTML events:**

| **Event** | **Description** |
| --- | --- |
| **onchange** | **An HTML element has been changed** |
| **onclick** | **The user clicks an HTML element** |
| **onmouseover** | **The user moves the mouse over an HTML element** |
| **onmouseout** | **The user moves the mouse away from an HTML element** |
| **onkeydown** | **The user pushes a keyboard key** |
| **onload** | **The browser has finished loading the page** |

**For complete list of events:**

[**https://www.w3schools.com/jsref/dom\_obj\_event.asp**](https://www.w3schools.com/jsref/dom_obj_event.asp)

**—-------js-demo7.html—------------------------------------------**

<!DOCTYPE html>

<html>

<head>

<script>

function handleClick(){

document.getElementById('demo').innerHTML= Date();

}

</script>

</head>

<body>

<h1>JavaScript HTML Events</h1>

<h2>The onclick Attribute</h2>

**<button onclick="handleClick()">The time is?</button>**

<p id="demo">This is placeholder</p>

</body>

</html>

—----------------------------------------------------------------------------------

## **JavaScript Event Handlers**

Event handlers can be used to handle and verify user input, user actions, and browser actions

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="utf-8">**

**<title>JavaScript Example: Events onload, onmouseover and onmouseout</title>**

**<script>**

**var msgLoad = "Hello!";**

**</script>**

**</head>**

**<body onload="alert(msgLoad)">**

**<p>"Hello" alert Box appears <em>after</em> the page is loaded.</p>**

**<p onclick='this.style.color="blue"' onmouseover="this.style.color='red'"**

**onmouseout="this.style.color=''">Point your mouse pointer here!!!</p>**

**</body>**

**</html>**

**—----------------------------------------js-demo9.html—-----------**

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="utf-8">**

**<title>JavaScript Example: Separating HTML, CSS and JavaScript</title>**

**<link rel="stylesheet" href="../css/demo9.css">**

**</head>**

**<body>**

**<p>"Hello" alert Box appears <em>after</em> the page is loaded.</p>**

**<p id="magic">Point your mouse pointer here!!!</p>**

**<script src="../js/demo9.js"></script>**

**</body>**

**</html>**

**—----------------------------------demo9.css—----------**

**.highlightMouseover {**

**color: red;**

**}**

**.highlightClick {**

**color: powderblue;**

**}**

**—--------------------------demo9.js—---------------------------------------------window.onload = function() {**

**init();**

**alert("Hello!");**

**}**

**function init() {**

**document.getElementById("magic").onmouseover = function() {**

**this.className = "highlightMouseover";**

**}**

**document.getElementById("magic").onmouseout = function() {**

**this.className = "";**

**}**

**document.getElementById("magic").onclick = function() {**

**this.className = "highlightClick";**

**}**

**}**

**—----------------------------------------------------------------------**

We can select HTML element(s) within the current page via these JavaScript built-in functions:

1. document.getElementById(*idName*): returns the HTML element with id="*idName*", or null if the id does not exist. The id value should be unique within an HTML document.
2. document.getElementsByTagName(*tagName*): returns an array of HTML elements with the given HTML tag.
3. document.getElementsByClassName(*className*): returns an array of HTML elements having attribute class="*className*".
4. document.getElementsByName(*name*): returns an array of HTML elements having attribute name="*name*".

—-------------------------for .. of loop--------------------------------------

for (variable of iterable) {

// code block to be executed

}

{

const list = [1,2,3,4,5];

for(let i of list){

console.log(i);

}

}

—------------js-demo10.html—------------------------------------

**<!DOCTYPE html>**

**<html lang="en">**

**<head>**

**<meta charset="utf-8">**

**<title>JavaScript Example: Modifying the Text Content of HTML Elements</title>**

**</head>**

**<body>**

**<h1 id="heading1">Heading 1</h1>**

**<h2>Heading 2</h2>**

**<h2>Heading 2</h2>**

**<p class="para">Paragraph 1</p>**

**<p class="para">Paragraph 2</p>**

**<p class="para">Paragraph 3</p>**

**<input type="button" id="btn1" value="Change Heading 1" />**

**<input type="button" id="btn2" value="Change Heading 2" />**

**<input type="button" id="btn3" value="Change Paragraph" />**

**<script src="JSExInnerHtml.js"></script>**

**</body>**

**</html>**

**—--------------------------------------------------demo10.js—--------------------**

**window.onload = init;**

**function init() {**

**document.getElementById("btn1").onclick = changeHeading1;**

**document.getElementById("btn2").onclick = changeHeading2;**

**document.getElementById("btn3").onclick = changeParagraph;**

**}**

**function changeHeading1() {**

**//document.getElementById("heading1").innerHTML = "hello";**

**//same as given below, the below approach is recommended**

**//replace innerHTML with textContent**

**const elm = document.getElementById("heading1"); // id is unique**

**elm.textContent = "hello";**

**// Create a new element and insert after h1**

**const newElm = document.createElement('p');**

**newElm.textContent = "hello, hello";**

**elm.after(newElm);**

**}**

**function changeHeading2() {**

**const elms = document.getElementsByTagName("h2"); // Array of elements**

**for (let i = 0; i < elms.length; i++) { // using traditional for(;;) loop**

**elms[i].textContent = "hello, again";**

**}**

**}**

**function changeParagraph() {**

**const elms = document.getElementsByClassName("para"); // Array of elements**

**for (const elm of elms) { // using JavaScript's for...of loop**

**// using const-declaration as a new block-scope elm is created for each iteration**

**elm.textContent = "hello, again and again";**

**}**

**}**

**—------------------------------------------------------------------------------**

# **JavaScript HTML DOM EventListener**

### **Example**

Add an event listener that fires when a user clicks a button:

document.getElementById("myBtn")

.addEventListener("click", displayDate);

is same

document.getElementById("myBtn").onclick = displayDate;

**Assignment:**

Create a basic calculator that takes two numbers and displays the result. The basic operations

1. Add b. Subtract c. Multiply 4. divide

Create an html file, css and JS file and link them.

—----------------------------------------------------------------------------------------------------------------------js-demo11.html-----------------------------

#### **Intercepting a Hyperlink**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>JavaScript Example: Intercept an Hyperlink</title>

</head>

<body>

<h4>Interceptin Hyperlink</h4>

<a href="js-demo1.html" id="myLink">JavaScript Example 1</a>

<h5>ThanQ</h5>

<script src="../js/demo11.js" ></script>

</body>

</html>

—------------------demo11.js—---------------------------

window.onload = init;

function init() {

document.getElementById("myLink").onclick = showWarning;

}

function showWarning() {

return confirm("Warning! Proceed with care!");

}

—---------------------------------------------------------------------

**Javascript Functions**

1. Named functions
2. Anonymous functions
3. IIFE functions
4. Arrow functions

Higher-order functions:

1. Callback functions
2. Function returning Function
3. Closures
4. **Named Functions**

**function functionName([<parameter(s)>]){**

**}**

**{**

**//Named Functions**

**function sayHello(){**

**return "Hello";**

**}**

**function sayHi(name){**

**return "Hi, "+name;**

**}**

**const greet = sayHello();**

**console.log(greet);**

**console.log(sayHi("Ravi") );**

**}**

—---------------------------------------------------------------------------

The function arguments/parameters can have default values

**function functionName(parameterName=value){**

**}**

**We can call the function passing no value or a value.**

**{**

**function sayHi(name="Smith"){**

**return "Hi, "+name;**

**}**

**console.log(sayHi("Ravi") );**

**console.log(sayHi());**

**}**

**—-----------------------------------------------------------------------**

**Function Hoisting**

**We can all the named functions before its definition, is**

**called function hoisting.**

**{**

**console.log(sayHi("Ravi") );**

**console.log(sayHi());**

**function sayHi(name="Smith"){**

**return "Hi, "+name;**

**}**

**}**

**Note: Function hoisting applicable only for named functions**

**—--------------------------------------------------------------**

1. **Anonymous Function**

**A function without a name is called anonymous function.**

**Note: Function hoisting not allowed for anonymous functions.**

**const functionRef = function([<parameters>]){**

**}**

**{**

**//Anonymous function**

**const funRef = function(firstName, lastName){**

**let fullName = (firstName+" "+ lastName).toUpperCase();**

**return fullName;**

**}**

**const name = funRef("Ravi","Kumar");**

**console.log(name);**

**}**

**4. IIFE : Immediately Invoke Function Expression**

**In this method, the function is not only defined but executed at same time.**

**(function(){**

**})();**

**{**

**const name= (function(firstName, lastName){**

**let fullName = (firstName+" "+ lastName).toUpperCase();**

**return fullName;**

**})("Ravi","Kumar");**

**console.log(name);**

**}**

**5. Arrow function**

**Introduce in ES6 and is short-cut verion of defining a function.**

**const funRef = ([<parameters>]) => {**

**};**

**{**

**const funRef = () => {**

**console.log("Hi, I'm a arrow function");**

**};**

**funRef();**

**const funRef1 = (fn,ln) => {**

**let fullName= (fn+" "+ ln).toUpperCase();**

**return fullName;**

**};**

**console.log(funRef1("Ravi","Kumar"));**

**const funRef2 = (fn,ln) => {**

**return (fn+" "+ ln).toUpperCase();**

**};**

**console.log(funRef2("Ravi","Kumar"));**

**//same as**

**const funRef3 = (fn,ln) => (fn+" "+ ln).toUpperCase();**

**console.log(funRef3("Ravi","Kumar"));**

**console.log( (()=>"Hello" )());**

**}**

**{**

**const f = (nm) => "Hi, "+nm ;**

**console.log(f("Smith"));**

**console.log( (()=>"Hello" )());**

**}**

**II. Higher-order Functions**

1. **Callback function**

**A function that is passed as an argument to another function is called callback function.**

**It is called so because the receiving function invokes the callback function when it requires.**

**receivingFunction( cb ){**

**—-----------—-----;**

**cb(); //invoking callback function**

**—--------------------;**

**}**

**//callback function**

**function f(){**

**//task**

**}**

**//passing reference of function f()**

**receivingFunction( f );**

**—---------------------------------------------**

**{**

**function fun1(){**

**return "hello";**

**}**

**const fun2 = function(){**

**return "hello";**

**}**

**const fun3 = () => "hello";**

**//cb is called formal parameter**

**function fun(cb){**

**console.log("I'm receving a callback function");**

**let m = cb();//invoking the callback function**

**console.log(m);**

**}**

**//fun1,fun2,fun3 are called as actual parameters**

**//here actual parameters are callback functions**

**fun(fun1);**

**fun(fun2);**

**fun(fun3);**

**//fun( fun3() ); //invoking fun3() and its response is sent to fun()**

**}**

**—--------------------------------------------**

**{**

**const f1 = ( cb) =>{**

**return cb() + " again";**

**}**

**// const f2 = () => "Hello";**

**//f1(f2);**

**//same as**

**const m = f1( () => "Hello" );**

**console.log(m);**

**}**

**—--------------------------------------------------------------**

1. **Function returning a function**

**const f = () => {**

**—------------;**

**return () => { } ;**

**}**

**const f1 = f()**

**f1();**

**—--------------------------------------------------------------**

**{**

**//function returning a function**

**const f1 = (m) => {**

**const greet = "Hi, "+ m;**

**console.log(greet);**

**return ()=> greet; //returning arrow function**

**};**

**const f2 = f1("Ravi"); //Hi, Ravi**

**console.log( f2() ); //Hi, Ravi**

**}**

**The inner function is returning a variable that belongs to the outer function.**

**When the inner function is executed, it is able to access the variables even after the execution of outer function.**

**This concept is called as closures.**

**Even after execution of outer function, the inner is able to access variables that belong to outer function.**

**In the above example, f1() is the outer function and f2() is the inner function.**

**—------------------------------------------------------------**

**{**

**//function that receives a callback function and returns a function**

**const f1 = (cb) => {**

**const x = cb(); //invoking f2()**

**//const f = () => x;**

**//return f;**

**// the above 2 statements are same as following statement**

**return () => x; //returning inner function**

**}**

**const f2 = () => "Hello";**

**const f3 = f1( f2 );**

**console.log( f3() );**

**}**

**—--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------**

**Javascript : Asynchronous Vs Synchronous, Callbacks, Promises and async/await**

**Asynchronous Vs Synchronous**

<https://www.freecodecamp.org/news/asynchronous-javascript-explained/>

Execution of the statements sequentially is synchronous coding where the next statement gets executed only after execution of previous statement.

All the scrips written so far are synchronous.

**In Asynchronous programming, the long running tasks will not block the execution of remaining tasks.**

**{**

**const fun1 = () => { —------}**

**funi1();**

**const fun2 = () => {**

**//fetch data from remote server**

**}**

**// long-running task**

**fun2();**

**const fun3 = () => { —------}**

**funi3();**

**}**

By default, JavaScript is a synchronous, single threaded programming language. This means that instructions can only run one after another, and not in parallel.

To further understand the asynchronous nature of JavaScript, we will go through **callback functions, promises, and async and await.**

**Callback Function**

**Callbacks are those functions which are passed as arguments to another function and are executed when a particular task is completed.**

**{**

**console.log('fired first');**

**console.log('fired second');**

**//setTimeout() is an Javascript asynchronous function that //takes a callback function and**

**//time in milliseconds as parameters. The callback function is //executed after a delay of ateast 2 seconds**

**setTimeout(()=>{**

**console.log('fired third');**

**},2000);**

**console.log('fired last');**

**}**

**Execution of long-running tasks can be sent as callback functions to setTimeout() function to achieve asynchronous programming.**

**{**

**function fetchData(callback) {**

**setTimeout(() => {**

**callback('Data fetched');**

**}, 2000);**

**}**

**fetchData((result) => {**

**console.log(result); // Output: Data fetched**

**});**

**console.log("main code section");**

**}**

**This method was very efficient, but only to a certain point. Sometimes, developers have to make multiple calls to different sources in their code. In order to make these calls, callbacks are being nested until they become very hard to read or maintain. This is referred to as Callback Hell**

**Advantages of Callbacks**

* **They are very simple to use and are widely supported.**
* **Efficient when working with simple asynchronous operations.**

**Disadvantages of Callbacks**

* **Quite complicated when dealing with multiple asynchronous operations which leads to callback hell.**
* **Error handling is challenging task as the code becomes complicated and hard to understand.**

**To fix this problem, promises were introduced.**

# **Promises**

**Promises is a representation of completion or failure of any asynchronous operation. It allows chaining of multiple asynchronous operations. There are basically 3 states in promises i.e., resolve, pending and reject.**

**{**

**function fetchData() {**

**return new Promise((resolve, reject) => {**

**setTimeout(() => {**

**resolve('Data fetched');**

**}, 1000);**

**});**

**}**

**fetchData()**

**.then((result) => {**

**console.log(result); // Output: Data fetched**

**})**

**.catch((error) => {**

**console.error(error);**

**});**

**}**

**In the above code, we have a fetchData() function which returns a promise. If the promise is resolved, the result will be displayed and if it is rejected then the catch block will be executed which will display the error.**

**Chaining Promise:**

**fetchData()**

**.then((result) => {**

**return processData(result);**

**})**

**.then((processedData) => {**

**console.log(processedData);**

**})**

**.catch((error) => {**

**console.error(error);**

**});**

**In the above code, fetchData() function is either resolved or rejected. If the request is resolved, .then will be executed step-by-step. But is the request is rejected then the chain will be executed till it reaches the .catch block to display the error.**

**Advantages of Promises**

* **Promises solves the main problem of callback hell by providing chaining. This makes code more readable and clean.**
* **Error handling is improved with the help of promises as we can use .catch() for error handling in promises.**

**Disadvantages of Promises**

* **It requires deep understanding of Promises API as it includes multiple properties and methods.**

# **Async/Await**

**Async/await is a feature that is built on top of promises to make it better and efficient. It is more concise and provides a synchronous-like way to write asynchronous programs. await keyword is always used inside the async function scope.**

**{**

**async function fetchData() {**

**return new Promise((resolve, reject) => {**

**setTimeout(() => {**

**resolve('Data fetched');**

**}, 1000);**

**});**

**}**

**async function getData() {**

**try {**

**const result = await fetchData();**

**console.log(result); // Output: Data fetched**

**} catch (error) {**

**console.error(error);**

**}**

**}**

**getData();**

**}**

**In the above code, fetchData() function returns a promise. getData() is a async function which contains a try catch block . We have result which waits for the fetchData() function to get the result. If the promise is resolved then the result will be displayed otherwise catch block will be executed.**

**Advantages of Async/Await**

* **It is much more readable as compared to promises and callbacks. It is much alike synchronous code which is easier to understand.**
* **It is built on top of the promises which provides compatibility between the two.**

**Disadvantages of Async/Await**

* **It has limited support in the older versions.**

[**https://medium.com/womenintechnology/callbacks-vs-promises-vs-async-await-detailed-comparison-d1f6ae7c778a**](https://medium.com/womenintechnology/callbacks-vs-promises-vs-async-await-detailed-comparison-d1f6ae7c778a)

**—-----------------------------------------------------------------------**

**AJAX**

**Asynchronous Javascript and XML**

Asynchronous JavaScript and XML (AJAX) is a combination of web application development technologies that make web applications more responsive to user interaction.

Whenever your users interact with a web application, such as when they click buttons or checkmark boxes, the browser exchanges data with the remote server.

Data exchange can cause pages to reload and interrupt the user experience. With AJAX, web applications can send and receive data in the background so that only small portions of the page refresh as required.

<https://aws.amazon.com/what-is/ajax/>

<https://www.loginradius.com/blog/engineering/ajax-and-xhr-using-plain-javascript/>

AJAX is a developer's dream, because you can:

* Update portion/part of a web page without reloading the entire page
* Request data from a server - after the page has loaded
* Receive data from a server - after the page has loaded
* Send data to a server - in the background

**AJAX Use Cases**

### Autocomplete

Search engines provide autocomplete options in real time when users search for a specific keyword in the search bar. AJAX allows the webpage to relay each character input to the web server and return a list of relevant recommendations on the existing page.

### Form verification

AJAX allows web applications to validate specific information in forms before users submit them. For example, when a new user creates an account, the webpage can automatically verify if a username is available before the user moves to the next section.

### Chat functionality

Text messengers and chatbots use AJAX to display real-time conversations on browsers. AJAX sends the text written by a user to the server and publishes it simultaneously in other users' chat interfaces.

### Social media

Social media platforms use AJAX to update users' feeds with the latest content without loading a new page on the browser. For example, Twitter refreshes your feed immediately whenever someone you follow tweets an update.

Ajax stands for Asynchronous Javascript and XML. Ajax is a programming technique that allows us to create dynamic, complex, and asynchronous web applications.

Ajax allows us to send and receive data from the webserver asynchronously without interfering with the current state or behavior of the web page or application.

XHR is the XMLHttpRequest Object which interacts with the server.

Ajax technique in the nutshell leverages the XHR request to send and receive data from the webserver.

This object is provided by the browser’s javascript environment. It transfers the data between the web browser and server.

#### **Key technologies for incorporating AJAX -**

* HTML DOM(document object model)
* JSON/XML
* XMLHttpRequest
* Javascript

### **Sending an XHR request**

To send and receive data from the server and implement the Ajax simple steps are explained below:

* Create a XMLHttpRequest object.
* Send the request to retrieve data from the server.
* Receive the response and display information to the end-user.

**Create XHR object**

var xhrobj = new XMLHttpRequest();

**Send the XHR object**

xhrobj.open('GET','[example.com/get](http://example.com/get)');

xhrobj.send();

|  |  |
| --- | --- |
| open(method, url[, async[, user[, password]]]) | It initializes the request. |
| method | request type such as GET,POST etc |
| url | Request URL |
| Async | true or false |
| user | Username for basic authentication |
| password | Password for basic authentication |
| send(body) | It sends the request to the server body : it is optional to send body of data with request. |

#### **3. Receiving the response :**

On completion of the request, the server sends the response to the request

**xhrobj.onreadystatechange = function () {**

**if (this.readyState == 4 && this.status == 200) {**

**document.getElementById("response").innerHTML = xhrobj.responseText;**

**}**

**}**

| onreadystatechange = callback() | It is a EventHandler called when the readyState attribute changes. |
| --- | --- |
| readyState attribute | It is an attribute that returns the current state of XMLHttpRequest object |
| status attribute | It is an attribute that returns the status code to the HTTP XHR request. |
| responseText attribute | It is an attribute that returns the DOMstring response as the text. |

| **Property** | **Description** |
| --- | --- |
| onreadystatechange | Defines a function to be called when the readyState property changes |
| readyState | Holds the status of the XMLHttpRequest.  0: request not initialized  1: server connection established  2: request received  3: processing request  4: request finished and response is ready |
| status | 200: "OK"  403: "Forbidden"  404: "Page not found"  For a complete list go to the [Http Messages Reference](https://www.w3schools.com/tags/ref_httpmessages.asp) |
| statusText | Returns the status-text (e.g. "OK" or "Not Found") |

—----------------------------------------------------------------------------------------------------------------------------------------------

<https://jsonplaceholder.typicode.com>

**Sample JSON object**

**{**

**"userId": 1,**

**"id": 1,**

**"title": "delectus aut autem",**

**"completed": false**

**}**

**Javascript object of the the above is:**

**{**

**userId: 1,**

**id: 1,**

**title: "delectus aut autem",**

**completed: false**

**}**

—------------------------------------------------------------------------------------ajax-demo.html—------------------------------------

<!DOCTYPE html>

<html>

<head>

<script src="https://ajax.googleapis.com/ajax/libs/jquery/2.1.1/jquery.min.js"></script>

<script src="../js/ajax-demo.js"></script>

</head>

<body>

<div>Ajax Application</div>

<div style="background-color:powderblue;">

Asynchronous JavaScript and XML (AJAX) is a combination of web application <br>

development technologies that make web applications more responsive to user interaction. <br>

Whenever your users interact with a web application, such as when they click buttons or checkmark <br>

boxes, the browser exchanges data with the remote server. Data exchange can cause pages to <br>

reload and interrupt the user experience. With AJAX, web applications can send and receive data<br>

in the background so that only small portions of the page refresh as required.

</div>

<div style="background-color:rgb(175, 174, 120);" id="divId">This portion will be changed</div>

<div class="append-to-me"></div>

<div style="background-color:rgb(218, 233, 245);">

You can use AJAX to create various features in web applications.

Autocomplete

Search engines provide autocomplete options in real time when users search for a <br>

specific keyword in the search bar. AJAX allows the webpage to relay each character <br>

input to the web server and return a list of relevant recommendations on the existing page.

<br>Form verification

<br>AJAX allows web applications to validate specific information in forms before users

<br>submit them. For example, when a new user creates an account, the webpage can

<br>automatically verify if a username is available before the user moves to the next section.

</div>

<button type="button" onclick="loadDoc1()">Get External Content</button>

<!-- <button>Get External Content</button> -->

</body>

</html>

—----------------ajax-demo.js—------------

function loadDoc1() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById("divId").textContent =

this.responseText;

}

};

xhttp.open("GET", "https://jsonplaceholder.typicode.com/posts/1", true);

xhttp.send();

}

function loadDoc2() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

document.getElementById("divId").textContent =

this.responseText;

}

};

xhttp.open("GET", "https://jsonplaceholder.typicode.com/posts/2", true);

xhttp.send();

}

// $(document).ready(function(){

// $("button").click(function(){

// $.ajax({

// type: 'GET',

// url: 'https://jsonplaceholder.typicode.com/posts/1',

// // url: 'https://jsonplaceholder.typicode.com/posts',

// success: function(response) {

// // if your response is in JSON format, you can access response keys

// // in the following format (both ways give same result)

// var title = response.title; // OR var title = response['userId']);

// // ...and then append to the DOM (your HTML) by selecting the

// // element you want to append to, and using the append method

// $('div.append-to-me').append(title);

// // $('div.append-to-me').append(response);

// },

// error: function(error) {

// console.log('not implemented');

// }

// });

// });

// });

—--------------------------------------------------------------------

# **JavaScript Classes**

ES6, introduced JavaScript Classes. JavaScript Classes are templates for JavaScript Objects.

class ClassName {

constructor() { ... }

}

Ex.

**{**

**class Car {**

**constructor(name, year) {**

**this.name = name;**

**this.year = year;**

**}**

**age(x) {**

**return x - this.year;**

**}**

**getName(){**

**return this.name;**

**}**

**}**

**const myCar = new Car("Ford", 2014);**

**const date = new Date();**

**let year = date.getFullYear();**

**console.log("My Car,"+myCar.getName()+" is "+ myCar.age(year) +" years old");**

**}**

—---------------------------------------------------------------------

**React JS**

[**https://react.dev/**](https://react.dev/)

[**https://legacy.reactjs.org/tutorial/tutorial.html#overview**](https://legacy.reactjs.org/tutorial/tutorial.html#overview)

React is a declarative, efficient, and flexible JavaScript library for building user interfaces. It lets you compose complex UIs from small and isolated pieces of code called “components”.

React components can be be created as class components or as functional components.

Functional components are preferred over class components by the developer community.

# **Introduction to React Js**

<https://taglineinfotechus.medium.com/why-is-react-js-called-as-single-page-application-87864595d2ba>

React.js, also known as React, is a popular open-source JavaScript library developed and maintained by Facebook. It is primarily used for building user interfaces and has become one of the most widely adopted front-end frameworks in the web development industry. React.js is known for its component-based architecture, which allows developers to create reusable UI components and efficiently manage the state of these components.

React.js is often associated with the concept of Single Page Applications (SPAs), which is a web application model that loads a single HTML page and dynamically updates that page as the user interacts with the app. This approach is in contrast to traditional multi-page web applications, where each user action triggers a full page reload.

# **Understanding Single Page Applications**

A Single Page Application (SPA) is a web application that loads a single HTML page and dynamically updates that page as the user interacts with the app. Instead of the traditional approach of loading a new page from the server for each user action, an SPA uses JavaScript to update the current page, providing a seamless and efficient user experience.

In an SPA, the initial page load fetches all the necessary resources (HTML, CSS, JavaScript, and data) required for the application to function. Subsequent user interactions are handled by the client-side JavaScript, which communicates with the server through APIs to fetch or update data, without requiring a full page reload.

# **Features of React Js that make it ideal for Single Page Applications**

React.js is particularly well-suited for building Single Page Applications due to several key features:

1. **Component-based Architecture**: React’s component-based approach allows developers to break down the user interface into reusable, modular pieces. This makes it easier to manage the complexity of large-scale applications and facilitates efficient state management.
2. **Virtual DOM**: React uses a Virtual DOM (Document Object Model) to efficiently update the actual DOM, minimizing the number of DOM manipulations required. This improves the overall performance of the application, which is crucial for a smooth user experience in an SPA.
3. **Declarative Programming**: React’s declarative programming model simplifies the process of building and updating the user interface. Developers can focus on describing the desired UI state, and React will handle the necessary updates.
4. **Unidirectional Data Flow**: React’s unidirectional data flow, where data flows from parent to child components, promotes a predictable and maintainable application structure, making it well-suited for building complex SPAs.
5. **Efficient Routing**: React provides robust routing solutions, such as React Router, which allow developers to implement client-side routing and navigation within an SPA, without the need for full page refreshes.
6. **Ecosystem and Community**: React has a thriving ecosystem of libraries, tools, and community support, which makes it easier to build and scale SPA applications with React.js.

**Install NodeJS**

**sudo apt install nodejs**

**Install NPM**

**sudo apt install npm**

**Uninstalling**

**sudo apt-get remove nodejs**

**sudo apt-get remove npm**

**sudo apt-get update**

**Install curl**

**To run uri at CLI**

**sudo apt install curl**

**# installs nvm (Node Version Manager)**

**curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash**

**Close the terminal and open a new terminal and execute the foll. command:**

**# download and install Node.js (you may need to restart the terminal)**

**nvm install 20**

**# verifies the right Node.js version is in the environment**

**node -v # should print `v20.16.0`**

**# verifies the right npm version is in the environment**

**npm -v # should print `10.8.1`**

**Insta**

Create folder, react, open the terminal in VSCode, go to the folder and at prompt

**Install create-react-app module**

**npm install create-react-app**

Create React application:

**npx create-react-app react-app**

Note: react-app is your project name

got to the folder react-app at the terminal

**cd react-app**

**Run the foll. at the terminal:**

**npm start**

**—-----------------------react-app—---------------------**

**App.js**

**—--------------------------------------**

**function App() {**

**return (**

**<div>**

**<h3>Welcome To My First React Application</h3>**

**<h5>ThanQ For Visiting My Site!</h5>**

**</div>**

**);**

**}**

**export default App;**

**—-------------index.js—---------------------------**

**import React from 'react';**

**import ReactDOM from 'react-dom/client';**

**import App from './App';**

**const root = ReactDOM.createRoot(document.getElementById('root'));**

**root.render(**

**<App />**

**);**

**—------------------------------------------App.js----------------------------------**

**//Functional Component**

**// function App(){**

**// return (**

**// <div>**

**// <h3>Welcome To My First React Application</h3>**

**// <h5>ThanQ For Visiting My Site!</h5>**

**// </div>**

**// );**

**// }**

**// export default App;**

**import React from 'react';**

**//Class Component**

**class App extends React.Component{**

**render(){**

**return (**

**<div>**

**<h4>Welcome to my page</h4>**

**</div>**

**);**

**}**

**}**

**export default App;**

**-----------------Standard Procedure---------------------------**

**//Functional Component**

**const App = () => {**

**return (**

**<div>**

**<h3>Welcome To My First React Application</h3>**

**<h5>ThanQ For Visiting My Site!</h5>**

**</div>**

**);**

**}**

**export default App;**

**—-------------------------------------App.js—--------------**

**//Functional Component**

**// const App = () => {**

**// return (**

**// <div>**

**// <h3>Welcome To My First React Application</h3>**

**// <h5>ThanQ For Visiting My Site!</h5>**

**// </div>**

**// );**

**// }**

**// export default App;**

**// import React from 'react';**

**//Class Component**

**// class App extends React.Component{**

**// render(){**

**// return (**

**// <div>**

**// <h4>Welcome to my page</h4>**

**// </div>**

**// );**

**// }**

**// }**

**// function sample1(){**

**// return <h1>Hello</h1>;**

**// }**

**// function sample2(){**

**// return <h1>Hello Again</h1>;**

**// }**

**// export default App;**

**// export {sample1,sample2};**

**//same as**

**// export {sample1};**

**// export {sample2};**

**—-----------------------props—---------------------------------**

**The parent component can send data to the child component and child component can receives the data as properties or shortly called as props.**

**The data flow in React is unidirectional i.e data flows from parent component to child component not otherwise.**

**Data flows from top to bottom in React component hierarchy.**

**props is always read-only, i.e the child component that receives props from the parent component cannot be modify them.**

**C1**

**C2**

**C3**

**If C1 has to send data to C3, it is not possible to bypass C2.**

**C1 —-data—>C2—--data—--->C3**

**import React from 'react';**

**// class App extends React.Component {**

**// render() {**

**// return (**

**// <ChildComponent name="First Child" />**

**// );**

**// }**

**// }**

**const App = () =>{**

**return (**

**<ChildComponent name="First Child" />**

**);**

**}**

**const ChildComponent = (props) => {**

**return <p>{props.name}</p>;**

**};**

**// class ChildComponent extends React.Component{**

**// render(){**

**// return <p>{this.props.name}</p>**

**// }**

**// }**

**export default App;**

**—-----------------------------------------------------------**

**// const ChildComponent = (props) => {**

**// return <p>Mr.{props.fname} {props.lname}</p>;**

**// };**

**const ChildComponent = ( {fname, lname }) => {**

**return <p> Mr.{fname} {lname}</p>;**

**};**

—-----------------------------------------------------------------------

Though a child component cannot send data to its parent component but can alter the data that belongs to parent component through event handlers.

The state of parent component can be altered by the child component. The current values in the variables of a component indicates its state.

—-----------------------------------------------------

By default functional components are stateless components i.e they do not state means cannot declare any variables in it. That is why functional component are called stateless components.

Class components have state i.e we can declare variables.

React has introduced React Hooks which provides state to functional components.

Hooks allow us to "hook" into React features such as state and lifecycle methods.

**The React useState Hook allows us to track state in a function component.**

State generally refers to data or properties that need to be tracking in an application.

**Array Destructuring**

**{**

**//Array destructuring**

**let arr= ["Welcome","To","My","React","program"];**

**//let greet = arr[0];**

**//let appl = arr[3];**

**let [greet,,,appl] = arr;**

**console.log(greet+","+appl);**

**let[greet1, ...temp] = arr;**

**console.log(greet1);//Welcome**

**console.log(temp);**

**}**

**{**

**//Array destructuring**

**function getArray() {**

**return ["Hello", "I" , "am", "Sarah"];**

**}**

**let [greeting,pronoun] = getArray();**

**console.log(greeting);//"Hello"**

**console.log(pronoun);//"I"**

**let [greetings = "hi",name = "Sarah"] = ["hello"];**

**console.log(greetings);//"Hello"**

**console.log(name);//"Sarah"**

**}**

**Object Destructuring**

{

//object destructuring

let person = {name: "Sarah", country: "Nigeria", job: "Developer"};

//let name = person.name;

//let country = person.country;

let { name,country,job } = person;

console.log(name+","+country+","+job);

}

**Note: The variable names have to be same as the property names of the JS object.**

**—-----------------------------------------------------------------------------------------------------------------------------**

**const App = ()=>{**

**const greeting = () =>{**

**alert("Welcome to React");**

**}**

**return(**

**<>**

**<button onClick = {greeting}>Click me</button>**

**</>**

**);**

**}**

**export default App;**

**—----------------------------------------------------------------------------------------**

**import { useState } from "react";**

**const App = () => {**

**const [counter,setCounter] = useState(0);**

**// const handleCounter = () => {**

**// setCounter(counter+1);**

**// }**

**return (**

**<div>**

**{/\* <button onClick={handleCounter} >Click Me</button> \*/}**

**<button onClick={() => { setCounter(counter+1);}} >Click Me</button>**

**<h4>{counter}</h4>**

**</div>**

**);**

**}**

**—---------------------------------------------------------**

**import { useState } from "react";**

**const App = () =>{**

**const [counter,setCounter] = useState(0);**

**return (**

**<div>**

**<ChildComponent onChangeC={ () => setCounter(counter+1)} />**

**<h4>{counter}</h4>**

**</div>**

**);**

**}**

**const ChildComponent = ({onChangeC}) =>{**

**return (**

**<div>**

**<button onClick={()=> onChangeC()}>Click Me </button>**

**</div>**

**)**

**}**

**export default App;**

**—-------------------------------App.js--------------------------**

**import { useState } from "react";**

**import Child from "./Child";**

**export default function App() {**

**let [state, setState] = useState("Initial");**

**// function handleState(newValue) {**

**// setState(newValue);**

**// }**

**return (**

**<div>**

**<h2>**

**Handling the <i> parent state from child component </i> in ReactJS.**

**</h2>**

**<div>**

**The input value in child state accessing from parent state is {state}.**

**</div>**

**<br></br>**

**{/\* pass handleState function as a prop of child component \*/}**

**{/\* <Child change = {handleState} /> \*/}**

**<Child onChangeC={ (newValue)=>{setState(newValue)} }/>**

**</div>**

**);**

**}**

**—-------------------------Child.js—----------------------------**

**import { useState } from "react";**

**// accessing the change function from the prop.**

**function Child({ onChangeC }) {**

**const [value, setValue] = useState("");**

**function handleChange(event) {**

**let value = event.target.value;**

**setValue(value);**

**onChangeC(value);**

**}**

**return (**

**<div>**

**<input**

**placeholder = "Enter some text"**

**value = {value}**

**onChange = {handleChange}**

**/>**

**</div>**

**);**

**}**

**export default Child;**

**—-----------------------------------------------------------------------------------------------------------**

**Understanding DOM and VIrtual DOM**

**https://code.pieces.app/blog/dom-virtual-dom-react**

DOM, or Real DOM, is an acronym for Document Object Model. It’s simply the UI of an application. When your application changes, the DOM updates to reflect the change. DOM represents the structure and content of a website in the form of a tree. With DOM, scripts and webpages can manipulate, modify and identify the components of a website.

### The Inefficiency of DOM

DOM is slow, unlike modern-day websites, which are fluid and reflect modifications almost immediately. DOM was intended for earlier websites, which were static because they were rendered by the server. By contrast, modern-day websites carry out rendering in the browser. Therefore, when using React DOM for modern websites, it’s slow. This is why Virtual DOM exists.

## **What is Virtual DOM?**

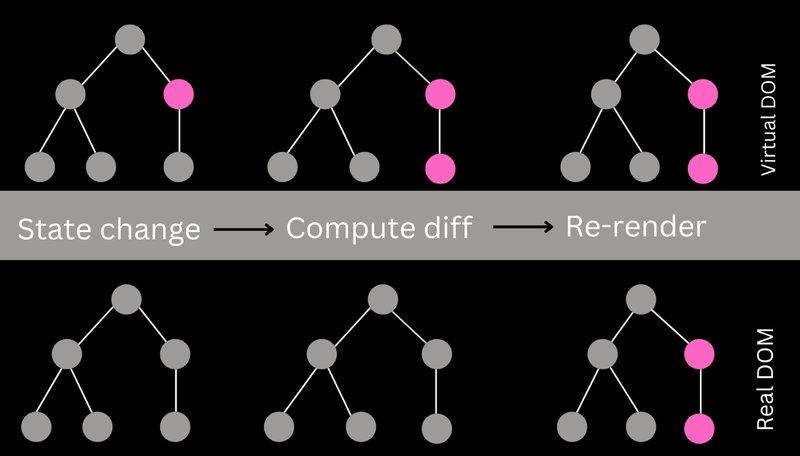
Virtual DOM is simply a copy of DOM. It has all the components and features of the DOM, but it cannot display the page of a website in a browser.

When there are a large number of elements in your DOM, it can be taxing to carry out updates using DOM. However, because of React Virtual DOM’s [observable pattern](https://en.m.wikipedia.org/wiki/Observer_pattern), it’s considerably faster.

### **Why is Virtual DOM Faster?**

When new elements are added to the UI, it creates a new Virtual DOM tree. On the tree, the elements are represented by colored spots. If any of the elements change, it creates a new tree. We then compare the new tree to the former tree to see which method is best to apply changes to the DOM.

Here's a visual representation of the explanation:

****

**As a result, the DOM does less work, which reduces the performance cost of updating it.**

## **How Does React Work with Virtual DOM?**

In React, each piece of UI is a component, and each component has a given state or condition. React pays attention to the [observable pattern](https://en.m.wikipedia.org/wiki/Observer_pattern) and to changes in state. When there is a change in a component, React immediately updates the Virtual DOM tree. After it updates, React compares the new tree with the former tree in a process called '[diffing](https://www.geeksforgeeks.org/what-is-diffing-algorithm/)'.

After identifying the changed tree, React updates the [objects](https://www.techtarget.com/searchapparchitecture/definition/object) that have been changed in the Real DOM.

### **React’s Rendering Function**

Render() is the process that updates and renders UI. The lifecycle method in React is render()*.*

The render() function creates the tree containing the React elements. When the state of a component is updated, render() provides a different tree containing the new React elements. By using setState() in the component, React immediately identifies the change in the state of the component and then re-renders it.

After that, React finds a way to update the UI so that the recent changes can be reflected. At this point, React updates its Virtual DOM, and in turn, updates the changed objects on the real DOM.

#### 

**—----------------------------------------------------------------------------------------------------------------------------------------------**

**React Component Lifecycle**

React Lifecycle is defined as the series of methods that are invoked in different stages of the component’s existence.

A React component undergoes the following phases:

### **Initialization phase**

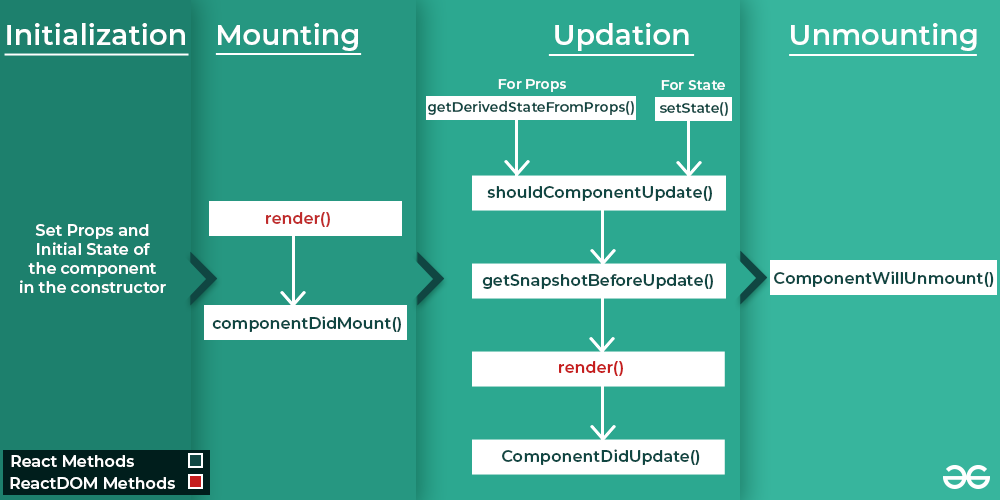
This is the stage where the component is constructed with the given Props and default state. This is done in the constructor of a Component Class. This is for class components.

### **2. Mounting Phase**

Mounting is the stage of rendering the JSX returned by the render method itself.

### **3. Updating**

Updating is the stage when the state of a component is updated and the application is repainted.



**React Hooks**

React Hooks provide [functional components](https://www.geeksforgeeks.org/reactjs-functional-components/) with the ability to use states and manage side effects.

They were first introduced in React 16.8, and allow developers to hook into the state and other React features without having to write a class.

They provide a cleaner and more concise way to handle state and side effects in React applications.

[**https://legacy.reactjs.org/docs/hooks-overview.html**](https://legacy.reactjs.org/docs/hooks-overview.html)

In React, side-effects are operations that interact with the external world, such as API calls, timers, manual DOM manipulations, or subscriptions. Given React’s declarative nature, managing side-effects requires specific patterns and tools to ensure smooth integration with the component lifecycle.

## **State Hook :** useState()

useState is a *Hook* that we call inside a function component to add some local state to it. React will preserve this state between re-renders.

useState returns a pair: the *current* state value and a function that lets you update it.

**function ExampleWithManyStates() {**

**// Declare multiple state variables!**

**const [age, setAge] = useState(42);**

**const [fruit, setFruit] = useState('banana');**

**const [todos, setTodos] = useState([{ text: 'Learn Hooks' }]);**

**// ...**

**}**

**Effect Hook: UseEffect()**

**The Effect Hook, useEffect, adds the ability to perform side effects from a function component. It serves the same purpose as componentDidMount, componentDidUpdate, and componentWillUnmount in React classes, but unified into a single API.**

In class components, developers typically manage side-effects within lifecycle methods. For instance, the `componentDidMount` method is a common place to initiate side-effects like fetching data, while `componentWillUnmount` is used to clean up, such as clearing timers or cancelling API requests.

However, the introduction of hooks in React brought a more unified and streamlined approach to handle side-effects in functional components using the `useEffect` hook.

The `useEffect` hook takes two arguments: a function containing the side-effect logic and an optional dependency array. The function runs after the component’s render, combining the roles of both `componentDidMount` and `componentDidUpdate` from class components.

The dependency array plays a vital role in optimizing the side-effect’s behavior. If the array is:

1. **Omitted:** The side-effect runs after every render.
2. **Empty (`[]`):** The side-effect runs once, similar to `componentDidMount`.
3. **Contains Values:** The side-effect runs only when the specified values change.

**import React, { useState, useEffect } from 'react';**

**function Example() {**

**const [count, setCount] = useState(0);**

**// Similar to componentDidMount and componentDidUpdate:**

**useEffect(() => {**

**// Update the document title using the browser API**

**document.title = `You clicked ${count} times`;**

**});**

**return (**

**<div>**

**<p>You clicked {count} times</p>**

**<button onClick={() => setCount(count + 1)}>Click me </button>**

**</div>**

**);**

**}**

## **Rules of Hooks**

Hooks are JavaScript functions, but they impose two additional rules:

* Only call Hooks at the top level. Don’t call Hooks inside loops, conditions, or nested functions.
* Only call Hooks from React function components. Don’t call Hooks from regular JavaScript functions.

**import React, { useState, useEffect } from 'react';**

**import axios from 'axios';**

**function App() {**

**const [count, setCount] = useState(0);**

**const fetchData = async () => {**

**try {**

**const response = await axios.get("https://api.github.com/users/mapbox");**

**console.log(response.data);**

**console.log(response.status);**

**console.log(response.statusText);**

**console.log(response.headers);**

**console.log(response.config);**

**} catch (error) {**

**// Handle error**

**console.error(error);**

**}**

**};**

**// Similar to componentDidMount and componentDidUpdate:**

**useEffect(() => {**

**// Update the document title using the browser API**

**// document.title = `You clicked ${count} times`;**

**fetchData();**

**},[count]);**

**return (**

**<div>**

**<p>You clicked {count} times</p>**

**<button onClick={() => setCount(count + 1)}>Click me </button>**

**</div>**

**);**

**}**

**export default App;**

**—-----------------------------------PersonsList.js-------------------------------**

**import { useState, useEffect } from 'react';**

**import axios from 'axios';**

**const PersonsList = () => {**

**const [persons, setPersons] = useState([]);**

**useEffect( async ()=> {**

**await axios.get(`https://jsonplaceholder.typicode.com/users`)**

**.then(res => {**

**setPersons(res.data);**

**})**

**},[]);**

**return (**

**<div>**

**<ul>**

**{persons.map((p) => (**

**<li key={Math.random().toString()}>**

**{p.id} {p.name}**

**</li>**

**))}**

**</ul>**

**</div>**

**);**

**}**

**export default PersonsList;**

**—------------------------------------------------App.js—--**

**import PersonsList from './PersonsList';**

**const App = () => {**

**return(**

**<PersonsList/>**

**);**

**}**

**export default App;**

**—-----------------------------------------------------------**

**Conditional Rendering**

In React, you can create distinct components that encapsulate behavior you need. Then, you can render only some of them, depending on the state of your application.

Conditional rendering in React works the same way conditions work in JavaScript. Use JavaScript operators like [if](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/if...else) or the [conditional operator](https://developer.mozilla.org/en/docs/Web/JavaScript/Reference/Operators/Conditional_Operator) to create elements representing the current state, and let React update the UI to match them.

Ex. We design buttons suitable for different layouts.We can choose one of the buttons for a particular layout based on some condition. This is called conditional rendering.

**import Goal from "./Goal";**

**const App = () => {**

**return(**

**<Goal isGoal={false}/>**

**);**

**}**

**export default App;**

**—--------------------------------------Goal.js—---------------------------**

**import React from 'react';**

**function MissedGoal() {**

**return <h1>MISSED!</h1>;**

**}**

**function MadeGoal() {**

**return <h1>GOAL!</h1>;**

**}**

**function Goal(props) {**

**const isGoal = props.isGoal;**

**console.log(isGoal);**

**if (isGoal === true) {**

**return <MadeGoal/>;**

**}**

**return <MissedGoal/>;**

**}**

**export default Goal;**

# **React Router v6**

[**https://code.pieces.app/blog/react-router-v6-a-comprehensive-guide-to-page-routing-in-react**](https://code.pieces.app/blog/react-router-v6-a-comprehensive-guide-to-page-routing-in-react)

Users can navigate through web pages due to a process called routing.

Routing in web applications is crucial, as it enables users to access different pages in an application upon request.

Since React does not provide its own router, install an external library called **React Router**.

React Router is [a library that provides the functionality](https://reactrouter.com/en/main) for client-side routing in React. It’s widely used in building [single-page applications (SPAs)](https://developer.mozilla.org/en-US/docs/Glossary/SPA).

With this library, users can navigate to pages on a website without triggering a page reload every time a new page is clicked. This also improves the speed and performance of an application.

## **Features of React Router**

The need for React Router and its importance expands beyond the navigation of web pages. It provides additional helpful features, some of which we will discuss in this section.

* **Client-side Routing**: Client-side routing allows users to access content from a web page without an additional server request. This removes the need for a page to reload as its content becomes available immediately after its link is clicked. This is a major upgrade to traditional websites where a page reload is initiated every time a user clicks a link.
* **Dynamic Routing**: React React Router adopts the dynamic approach of routing which allows routes to be defined during an application’s rendering state. This leads to faster load times and the development of more complex applications.
* **Redirects**: With React Router, you can conditionally navigate to a new route. Redirects allow users to navigate to a new location that overrides the history stack's current location. For example, a login page should redirect to a dashboard page after successful user authentication.
* **Error Handling**: React Router 6 handles most of the errors in your application, catching errors that are thrown while rendering, loading, or updating data.

**1. Install React Router**

**npm install react-router-dom@latest**

**2. Open index.js file and wrap your App component in the BrowserRouter component**

**import React from 'react';**

**import ReactDOM from 'react-dom/client';**

**import App from './App';**

**// import sample1 from './App';**

**import { BrowserRouter } from 'react-router-dom';**

**const root = ReactDOM.createRoot(document.getElementById('root'));**

**root.render(**

**<BrowserRouter>**

**<App />**

**</BrowserRouter>**

**);**

**3. Implementing Page Routing with React Router v6**

**—------------------------------Navbar.js—-------------------**

**import { Link } from "react-router-dom";**

**const Navbar = () => {**

**return (**

**<nav>**

**<ul>**

**<li>**

**<Link to="/">Home</Link>**

**</li>**

**<li>**

**<Link to="/about">About Pieces</Link>**

**</li>**

**<li>**

**<Link to="/snippet">Code Snippet</Link>**

**</li>**

**</ul>**

**</nav>**

**);**

**}**

**export default Navbar;**

**4. Create components**

**Home.js**

**—----------------------------------------**

**import React from 'react'**

**import Navbar from './Navbar'**

**const Home = () => {**

**return (**

**<div>**

**<Navbar />**

**<h3>Welcome to the Pieces Tutorial</h3>**

**</div>**

**)**

**}**

**export default Home;**

**About.js**

**—-------------------------------------------------**

**import Navbar from "./Navbar";**

**const About = () => {**

**return (**

**<div>**

**<Navbar />**

**<h3>**

**Pieces is your AI-enabled productivity tool designed to supercharge**

**developer efficiency**

**</h3>**

**<h3>Unify your entire toolchain with an on-device copilot</h3>**

**</div>**

**);**

**};**

**export default About;**

**Snippet.js**

**—------------------------------------------------------------**

**import Navbar from "./Navbar";**

**const Snippet = () => {**

**return (**

**<div>**

**<Navbar />**

**<h3>Code Snippet</h3>**

**<h3>Have a full explanation and use case of the code provided</h3>**

**<h3>Determine possibilities and limitations of code</h3>**

**<h3>Ask specific questions based on the code provided.</h3>**

**</div>**

**);**

**};**

**export default Snippet;**

**—----------------------App.js—-------------------------**

**import { Route, Routes } from 'react-router-dom';**

**import Home from './Home';**

**import About from './About';**

**import Snippet from './Snippet';**

**function App() {**

**return (**

**<div>**

**<Routes>**

**<Route path='/' element={<Home/>} />**

**<Route path='/about' element={<About/>} />**

**<Route path='/snippet' element={<Snippet/>} />**

**</Routes>**

**</div>**

**);**

**}**

**export default App;**

**----------------------------------------------------------**

### **Concept of passing parameters to routes**

Passing parameters to routes in a React router enables components to access the parameters in a route's path.

These parameters can be anything from a user object to a simple string or number, which can be very helpful when, for example, you want to pass data related to a specific user from one component to another.

Parameters, often referred to as **params**, are dynamic parts of the URL that can change and are set to a specific value when a particular route is matched.

In React Router, parameters can be used to capture values from the URL, which can then be used to further personalize or specify what to render in a particular view.

By passing parameters, we're making routes dynamic, allowing us to reuse the same component for different data based on the parameter value.

#### **The Relation Between Parameters and React Router**

When defined in a route, parameters allow React Router to understand which part of the URL should be captured as a dynamic argument.

These parameters can then be accessed in the routed components using the **props** passed by React Router. This way, parameters help in creating dynamic and personalized routes in a React application.

For example, if you have a user component that takes a user ID as a parameter, you can define the route path as "**/user/:userId"**.

Here ":userId" is a parameter, and its value can be accessed in the matched component as a prop - **props.match.params.userId**. This mechanism enables the creation of a detail view for each user in the application using the same User component.

Note:

**props.match.params.userId is deprecated instead apply useParams React Hook method**

import { useParams } from 'react-router-dom';

Const {userId} = useParams()

—--------------------Navbar.js—-----------------

import { Link } from "react-router-dom";

const Navbar = () => {

return (

<nav>

<ul>

<li>

<Link to="/">Home</Link>

</li>

<li>

<Link to="/about">About Pieces</Link>

</li>

<li>

<Link to="/snippet">Code Snippet</Link>

</li>

<li>

<Link to="/user">User Details</Link>

</li>

</ul>

</nav>

);

}

export default Navbar;

—--User.js—------------

**import Navbar from "./Navbar";**

**import { useParams } from "react-router-dom";**

**const User = (props) => {**

**const {userId} = useParams();**

**return (**

**<>**

**<Navbar/>**

**<div>User ID: {userId}</div>**

**</>**

**)**

**}**

**export default User;**

**—---------------App.js—--------------------**

**import { Route, Routes} from 'react-router-dom';**

**import Home from './Home';**

**import About from './About';**

**import Snippet from './Snippet';**

**import User from './User';**

**function App() {**

**return (**

**<div>**

**<Routes>**

**<Route path='/' element={<Home/>} />**

**<Route path='/about' element={<About/>} />**

**<Route path='/snippet' element={<Snippet/>} />**

**<Route path="/user/:userId" element={<User/>} />**

**</Routes>**

**</div>**

**);**

**}**

**Export default App;**

**=========================================App.js==========**

**import { Route } from "react-router-dom";**

**import UserProfile from "./UserProfile";**

**function App() {**

**return (**

**<div>**

**<Routes>**

**<Route path="/user/:userId" element={<UserProfile/>} />**

**</Routes>**

**</div>**

**);**

**}**

**export default App;**

**—--------------------------------UserProfile.js—--------------------------------**

**import { useEffect, useState } from 'react';**

**import axios from 'axios';**

**import { useParams } from 'react-router-dom';**

**function UserProfile(props) {**

**const [user, setUser] = useState(null);**

**const {userId} = useParams();**

**useEffect(() => {**

**axios.get(`https://jsonplaceholder.typicode.com/users/${userId}`)**

**.then(response => {**

**setUser(response.data);**

**});**

**}, [userId]);**

**return user ? (**

**<div>{user.name}</div>**

**) : (**

**<div>Loading...</div>**

**);**

**}**

**export default UserProfile;**

**—--------------------------------------------------------------------------React CRUD Application—---------------------------**

[**https://www.freecodecamp.org/news/how-to-perform-crud-operations-using-react/**](https://www.freecodecamp.org/news/how-to-perform-crud-operations-using-react/)

[**https://mockapi.io/**](https://mockapi.io/)

**npx create-react-app react-crud**

**cd react-crud**

**npm install semantic-ui-react semantic-ui-css**

**Add the following to index.js file**

**import 'semantic-ui-css/semantic.min.css'**

**App.css**

**—---------------------------------------------------**

**.main{**

**display: flex;**

**justify-content: center;**

**align-items: center;**

**height: 100vh;**

**}**

**@import url('https://fonts.googleapis.com/css2?family=Montserrat&display=swap');**

**.main-header{**

**font-family: 'Montserrat', sans-serif;**

**}**

**App.js**

**—----------------------------------------**

**import './App.css';**

**function App() {**

**return (**

**<div className="main">**

**<h2 className="main-header">React Crud Operations</h2>**

**</div>**

**);**

**}**

**export default App;**

**After creating account in** [**https://mockapi.io**](https://mockapi.io)**, create a project and add resource. You will get API end-point.**

**To create React form, go the foll website and type form in search bar**

[**https://react.semantic-ui.com/**](https://react.semantic-ui.com/)

**—------------create.js—----------------**

**import React, { useState } from 'react';**

**import { Button, Checkbox, Form } from 'semantic-ui-react'**

**export default function Create() {**

**const [firstName, setFirstName] = useState('');**

**const [lastName, setLastName] = useState('');**

**const [checkbox, setCheckbox] = useState(false);**

**return (**

**<div>**

**<Form className="create-form">**

**<Form.Field>**

**<label>First Name</label>**

**<input placeholder='First Name' onChange={(e) => setFirstName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<label>Last Name</label>**

**<input placeholder='Last Name' onChange={(e) => setLastName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<Checkbox label='I agree to the Terms and Conditions' onChange={(e) => setCheckbox(!checkbox)}/>**

**</Form.Field>**

**<Button type='submit'>Submit</Button>**

**</Form>**

**</div>**

**)**

**}**

**—------------------------------------create.js—----------------**

**import React, { useState } from 'react';**

**import { Button, Checkbox, Form } from 'semantic-ui-react'**

**import axios from 'axios';**

**export default function Create() {**

**const [firstName, setFirstName] = useState('');**

**const [lastName, setLastName] = useState('');**

**const [checkbox, setCheckbox] = useState(false);**

**// const postData = () => {**

**// console.log(firstName);**

**// console.log(lastName);**

**// console.log(checkbox);**

**// }**

**const postData = () => {**

**axios.post(`https://66a39ba544aa63704581df17.mockapi.io/fakeData`, {**

**firstName,**

**lastName,**

**checkbox**

**})**

**}**

**return (**

**<div>**

**<Form className="create-form">**

**<Form.Field>**

**<label>First Name</label>**

**<input placeholder='First Name' onChange={(e) => setFirstName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<label>Last Name</label>**

**<input placeholder='Last Name' onChange={(e) => setLastName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<Checkbox label='I agree to the Terms and Conditions' onChange={(e) => setCheckbox(!checkbox)}/>**

**</Form.Field>**

**<Button onClick={postData} type='submit'>Submit</Button>**

**</Form>**

**</div>**

**)**

**}**

As you can see, we are using axios.post. And inside axios.post, we have the API endpoint, which

we created earlier. Then, we have the form fields wrapped in curly brackets.

When we click Submit, this function will be called and it will post data to the API server.

Ex.

<https://mockapi.io/projects/66a39ba544aa63704581df18/fakeData>

## **Implement the Read and Update Operations**

To start the read operation, we need to create a Read Page. We also need the React Router

package to navigate to different pages.

Install the package using

npm i react-router-dom.

After it has been installed, import a few things from React Router:

import { BrowserRouter as Router, Route } from 'react-router-dom'

In our **App.js**, wrap the whole return into a Router.

This basically means that whatever is inside this Router will be able to use routing in React.

**import './App.css';**

**import Create from './components/create';**

**import { BrowserRouter as Router, Route } from 'react-router-dom’**

**function App() {**

**return (**

**<Router>**

**<div className="main">**

**<h2 className="main-header">React Crud Operations</h2>**

**<div>**

**<Create />**

**</div>**

**</div>**

**</Router>**

**);**

**}**

**export default App;**

Our App.js will look like the above now.

Replace the Create inside the return and add the following code:

**import './App.css';**

**import Create from './components/create';**

**import { BrowserRouter as Router, Route } from 'react-router-dom'**

**function App() {**

**return (**

**<Router>**

**<div className="main">**

**<h2 className="main-header">React Crud Operations</h2>**

**<div>**

**<Route exact path='/create' component={Create} />**

**</div>**

**</div>**

**</Router>**

**);**

**}**

**export default App;**

Here, we are using the Route component as Create. We have set the path of Create to '/create'.

So, if we go<http://localhost:3000/create>, we will see the create page.

Similarly, we need routes for read and update.

**import './App.css';**

**import Create from './components/create';**

**import Read from './components/read';**

**import Update from './components/update';**

**import { BrowserRouter as Router, Route } from 'react-router-dom'**

**function App() {**

**return (**

**<Router>**

**<div className="main">**

**<h2 className="main-header">React Crud Operations</h2>**

**<div>**

**<Route exact path='/create' component={Create} />**

**</div>**

**<div style={{ marginTop: 20 }}>**

**<Route exact path='/read' component={Read} />**

**</div>**

**<Route path='/update' component={Update} />**

**</div>**

**</Router>**

**);**

**}**

**export default App;**

### **The Read Operation**

For the Read operation, we will need a Table component.

So, head over to React Semantic UI and use a table from the library.

Now, let's send the GET Request to get the data from the API.

We need the data when our application loads. So, we are going to use the useEffect Hook.

**import React, { useEffect } from 'react';**

**useEffect(() => {**

**}, [])**

Create one state that will contain the incoming data. This will be an array.

**import React, { useEffect, useState } from 'react';**

**const [APIData, setAPIData] = useState([]);**

**useEffect(() => {**

**}, [])**

**//https://66a39ba544aa63704581df17.mockapi.io/fakeData**

**useEffect(() => {**

**axios.get(`https://66a39ba544aa63704581df17.mockapi.io/fakeData`)**

**.then((response) => {**

**setAPIData(response.data);**

**})**

**}, [])**

So, we are using axios.get to send the GET request to the API. Then, if the request is

fulfilled, we are setting the response data in our *APIData* state.

Now, let's map our Table rows according to the API Data.

We are going to use the Map function to do this

**<Table.Body>**

**{APIData.map((data) => {**

**return (**

**<Table.Row>**

**<Table.Cell>{data.firstName}</Table.Cell>**

**<Table.Cell>{data.lastName}</Table.Cell>**

**<Table.Cell>{data.checkbox ? 'Checked' : 'Unchecked'}</Table.Cell>**

**</Table.Row>**

**)})}**

**</Table.Body>**

**—--------------------Read.js—-----------------------------------------**

**import React from 'react';**

**import { Table } from 'semantic-ui-react'**

**import React, { useEffect, useState } from 'react';**

**export default function Read() {**

**const [APIData, setAPIData] = useState([]);**

**useEffect(() => {**

**axios.get(`https://66a39ba544aa63704581df17.mockapi.io/fakeData`)**

**.then((response) => {**

**setAPIData(response.data);**

**})**

**}, []);**

**return (**

**<div>**

**<Table singleLine>**

**<Table.Header>**

**<Table.Row>**

**<Table.HeaderCell>First Name</Table.HeaderCell>**

**<Table.HeaderCell>Last Name</Table.HeaderCell>**

**<Table.HeaderCell>Checked</Table.HeaderCell>**

**</Table.Row>**

**</Table.Header>**

**<Table.Body>**

**{APIData.map((data) => {**

**return (**

**<Table.Row>**

**<Table.Cell>{data.firstName}</Table.Cell>**

**<Table.Cell>{data.lastName}</Table.Cell>**

**<Table.Cell>{data.checkbox ? 'Checked' : 'Unchecked'}</Table.Cell>**

**</Table.Row>**

**)})}**

**</Table.Body>**

**</Table>**

**</div>**

**)**

**}**

Following is the updated version of Read.js file

import { Table, Button} from 'semantic-ui-react'

import React, { useEffect, useState } from 'react';

import axios from 'axios';

import {Link} from 'react-router-dom';

export default function Read() {

const [APIData, setAPIData] = useState([]);

useEffect(() => {

axios.get(`https://66a39ba544aa63704581df17.mockapi.io/fakeData`)

.then((response) => {

setAPIData(response.data);

})

}, []);

// const setData = (data) => {

// console.log(data);

// }

const setData = (data) => {

let { id, firstName, lastName, checkbox } = data;

localStorage.setItem('ID', id);

localStorage.setItem('First Name', firstName);

localStorage.setItem('Last Name', lastName);

localStorage.setItem('Checkbox Value', checkbox)

console.log(data);

}

return (

<div>

<Table singleLine>

<Table.Header>

<Table.Row>

<Table.HeaderCell>First Name</Table.HeaderCell>

<Table.HeaderCell>Last Name</Table.HeaderCell>

<Table.HeaderCell>Checked</Table.HeaderCell>

<Table.HeaderCell>Update</Table.HeaderCell>

</Table.Row>

</Table.Header>

<Table.Body>

{APIData.map((data) => {

return (

<Table.Row>

<Table.Cell>{data.firstName}</Table.Cell>

<Table.Cell>{data.lastName}</Table.Cell>

<Table.Cell>{data.checkbox ? 'Checked' : 'Unchecked'}</Table.Cell>

<Link to='/update'>

<Table.Cell> <Button onClick={() => setData(data)}>Update</Button></Table.Cell>

</Link>

</Table.Row>

)})}

</Table.Body>

</Table>

</div>

)

}

### **The Update Operation**

### Create one more header for Update and one column in the table row for an update button.

### Use the button from Semantic UI React.

**<Table.HeaderCell>Update</Table.HeaderCell>**

**<Table.Cell>**

**<Button>Update</Button>**

**</Table.Cell>**

### Now, when we click this button, we should be redirected to the update page.

### For that, we need **Link** from React Router.

### Import Link from React Router. And wrap the table cell for the update button into

### Link tags.

### 

**import { Link } from 'react-router-dom';**

**<Link to='/update'>**

**<Table.Cell>**

**<Button>Update</Button>**

**</Table.Cell>**

**</Link>**

So, if we click update button, we will be redirected to the update page.

In order to update the column data, we need their respective ID's, which comes

from the APIs.

Create a function called setData. Bind it to the Update button.

**<Button onClick={() => setData(data)}>Update</Button>**

Now, we need to pass the data as a parameter to the top function.

Let's set this data into the localStorage.

**const setData = (data) => {**

**let { id, firstName, lastName, checkbox } = data;**

**localStorage.setItem('ID', id);**

**localStorage.setItem('First Name', firstName);**

**localStorage.setItem('Last Name', lastName);**

**localStorage.setItem('Checkbox Value', checkbox)**

**}**

We are destructuring our data into id, firstName, lastName, and checkbox, and

then we are setting this data into local storage. You can use local storage to store

data locally in the browser.

Now, in the Update component, we need one form for the update operation.

Let's reuse the form from our Create component

**—--------------------Update.js—---------------------------------**

**import React, { useState } from 'react';**

**import { Button, Checkbox, Form } from 'semantic-ui-react'**

**import axios from 'axios';**

**export default function Update() {**

**const [firstName, setFirstName] = useState('');**

**const [lastName, setLastName] = useState('');**

**const [checkbox, setCheckbox] = useState(false);**

**return (**

**<div>**

**<Form className="create-form">**

**<Form.Field>**

**<label>First Name</label>**

**<input placeholder='First Name' onChange={(e) => setFirstName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<label>Last Name</label>**

**<input placeholder='Last Name' onChange={(e) => setLastName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<Checkbox label='I agree to the Terms and Conditions' onChange={(e) =>**

**setCheckbox(!checkbox)}/>**

**</Form.Field>**

**<Button type='submit'>Update</Button>**

**</Form>**

**</div>**

**)**

**}**

Create a useEffect hook in the Update component.

We will use it to get data we previously stored in Local Storage. Also, create one

more state for the ID field.

**const [id, setID] = useState(null);**

**useEffect(() => {**

**setID(localStorage.getItem('ID'))**

**setFirstName(localStorage.getItem('First Name'));**

**setLastName(localStorage.getItem('Last Name'));**

**setCheckbox(localStorage.getItem('Checkbox Value'))**

**}, []);**

Set the respective data according to your keys from Local Storage.

We need to set these values in form fields. It will automatically populate the fields

when the Update page loads.

Setting the form fields

**<Form className="create-form">**

**<Form.Field>**

**<label>First Name</label>**

**<input placeholder='First Name' value={firstName} onChange={(e) =>**

**setFirstName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<label>Last Name</label>**

**<input placeholder='Last Name' value={lastName} onChange={(e) =>**

**setLastName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<Checkbox label='I agree to the Terms and Conditions' checked={checkbox} onChange={(e) =>**

**setCheckbox(!checkbox)}/>**

**</Form.Field>**

**<Button type='submit'>Update</Button>**

**</Form>**

Now, if we click the Update button in Read Page, we will be redirected to the

update page, where we will see all the auto populated form data.

Now, let's create the Update request to update the data.

Create a function called updateAPIData. Inside this function, we are going to use

axios.put to send a PUT request that will update our data.

const updateAPIData = () => {

axios.put(`https://66a39ba544aa63704581df17.mockapi.io/fakeData/${id}`, {

firstName,

lastName,

checkbox

})

}

Here, you can see we are appending the API endpoint with an id field.

When we click the field in the table, its ID is getting stored into Local Storage. And

in the Update page, we are retrieving it. Then, we are storing that ID in the *id*

state.

After that, we pass the id to the endpoint. This allows us to update the field of

which we are passing the ID.

Bind the updateAPIData function to the Update button.

**<Button type='submit' onClick={updateAPIData}>Update</Button>**

Click the Update button in the table in Read page, change your last name, and

then click the Update button in the Update page.

### **The Delete Operation**

Add one more Button in the Read table, which we'll use for the Delete operation.

**<Table.Cell>**

**<Button onClick={() => onDelete(data.id)}>Delete</Button>**

**</Table.Cell>**

Create a function called onDelete, and bind this function to the Delete button.

This function will receive an ID parameter on the Delete button click.

We are going to use axios.delete to delete the respective columns.

**const onDelete = (id) => {**

**axios.delete(`**[**https://**](https://60fbca4591156a0017b4c8a7.mockapi.io/fakeData/$%7Bid)66a39ba544aa63704581df17[**.mockapi.io/fakeData/${id**](https://60fbca4591156a0017b4c8a7.mockapi.io/fakeData/$%7Bid)**}`**

**)**

**}**

Click the Delete button and check the API. You will see the data has been deleted.

We need to load the table data after it has been deleted.

So, create a function to load the API data.

**const getData = () => {**

**axios.get(`https://**66a39ba544aa63704581df17**.mockapi.io/fakeData`)**

**.then((getData) => {**

**setAPIData(getData.data);**

**})**

**}**

Now, in the onDelete function, we need to load the updated data after we delete a field.

**const onDelete = (id) => {**

**axios.delete(`https://**66a39ba544aa63704581df17**.mockapi.io/fakeData/${id}`)**

**.then(() => {**

**getData();**

**})**

**}**

So, now if we click Delete on any field, it will delete that field and refresh the table

automatically.

## **Some Improvements to our CRUD App**

So, when we post our data in the Create page, we are just getting the data in the

mock database. We need to redirect to the Read page when our data is created in

the Create page.

Note: In **react-router-dom v6** useHistory() is replaced by useNavigate()

Import useNavigate from React Router.

**import { useNavigate } from 'react-router';**

Create a variable called navigate with let, and set it to useNavigate():

**const navigate = useNavigate();**

**navigate('/home');**

Then, use the **navigate** function to push to the Read page just after the post

API gets called.

**const postData = () => {**

**axios.post(`https://**66a39ba544aa63704581df17**.mockapi.io/fakeData`, {**

**firstName,**

**lastName,**

**checkbox**

**}).then(() => {**

**navigate('/read')**

**})**

**}**

It will push to the Read page using the useHistory hook.

Do the same for the Update page.

—-----------------------Update.js—-----------------------

**import React, { useState, useEffect } from 'react';**

**import { Button, Checkbox, Form } from 'semantic-ui-react'**

**import axios from 'axios';**

**import { useNavigate} from 'react-router';**

**export default function Update() {**

**let navigate = useNavigate();**

**const [id, setID] = useState(null);**

**const [firstName, setFirstName] = useState('');**

**const [lastName, setLastName] = useState('');**

**const [checkbox, setCheckbox] = useState(false);**

**useEffect(() => {**

**setID(localStorage.getItem('ID'))**

**setFirstName(localStorage.getItem('First Name'));**

**setLastName(localStorage.getItem('Last Name'));**

**setCheckbox(localStorage.getItem('Checkbox Value'));**

**}, []);**

**const updateAPIData = () => {**

**axios.put(`https://**66a39ba544aa63704581df17**.mockapi.io/fakeData/${id}`, {**

**firstName,**

**lastName,**

**checkbox**

**}).then(() => {**

**navigate('/read')**

**})**

**}**

**return (**

**<div>**

**<Form className="create-form">**

**<Form.Field>**

**<label>First Name</label>**

**<input placeholder='First Name' value={firstName} onChange={(e) => setFirstName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<label>Last Name</label>**

**<input placeholder='Last Name' value={lastName} onChange={(e) => setLastName(e.target.value)}/>**

**</Form.Field>**

**<Form.Field>**

**<Checkbox label='I agree to the Terms and Conditions' checked={checkbox} onChange={() =>**

**setCheckbox(!checkbox)}/>**

**</Form.Field>**

**<Button type='submit' onClick={updateAPIData}>Update</Button>**

**</Form>**

**</div>**

**)**

**}**

**================================================================**