**Java**

**Classes**

* In Java top level classes cannot be declared as private; only nested classes can.
* toString() method belongs to the Object class in Java. Every class in Java extends the Object class.
* Classes encapsulate properties/variables and methods.
* Classes can extend a single class and implement one or more interfaces.

**Methods**

* Java allows *covariant return types*, which means you can vary your return type so long as you are returning a sub class of your specified return type.
* Method overloading – same method name but different type parameters. Referred to as compile-time polymorphism. Polymorphism means same name but different forms.
* Method overriding – where a subclass is allowed to override the behaviour of an existing super class method. It is also known as runtime polymorphism. Method overriding is done using the reference of an object or using the super keyword.
* Method overriding can be prevented using private, static or final keywords. Using final keyword generates a compile time error. Private and static execute only the base class methods.

E.g.,

class Base {

public final String version(){

where();

System.out.println(“1.0.0”);

}

public static String name(){

return(“Base”);

}

private void where(){

System.out.println(“inside the base class”);

}

}

class Derived extends Base{

//compilation error

public final String version(){

}

public static String name(){

return(“Derived”);

}

private void where(){

System.out.println(“inside the derived class”);

}

}

O/P:

Inside Base Class

1.0.0

Base

**Interfaces**

* Interfaces are used to implement multiple inheritance in Java, it enhances the concept of single inheritance.
* The access modifier is either public or default(package)
* The variables are public, static and final by default and cannot be changed.

**Syntax:** public interface <interface name> extends <interface1>, <interface2>

* An interface can extend more than one interface.
* A class can implement more than one interfaces.
* All methods in an interface are abstract by default.
* An interface cannot be instantiated. But it can be assigned an instantiated object of a class in which it is implemented.
* E.g., List list = new ArrayList<String>();

Here, List is an interface and ArrayList is a class.

Q & A

* Can we return a sub-class of the object instead of the object itself?

**Constructors**

* They are methods that have the same name as the class.
* The are executed when an object is initialized.
* Types – default, parameterized and copy constructor.
* If a parameterized constructor is implemented, then it is required that the default constructor be explicitly defined.
* E.g.,

class Complex{

int x, y;

Complex(){

System.out.println(“default”);

}

Complex(int x, int y){

System.out.println(x + “ i“ + y);

}

Complex( Complex c){

this.x = c.x;

this.y = c.y;

}

}

**Collections vs Collection**.

* Collections is a class used to sort and synchronize the collection elements.
* E.g.,

List list = new ArrayList<String>();

Collections.addAll(list, “Apple”, “Orange”, “Lime”);

Collections.sort(list);

Collections.sort(list, Comparator.reverseOrder());

Collections.binarySearch(list, “Orange”);

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

Collections.copy(list, fruits);

Collections.disjoint(list,fruits);

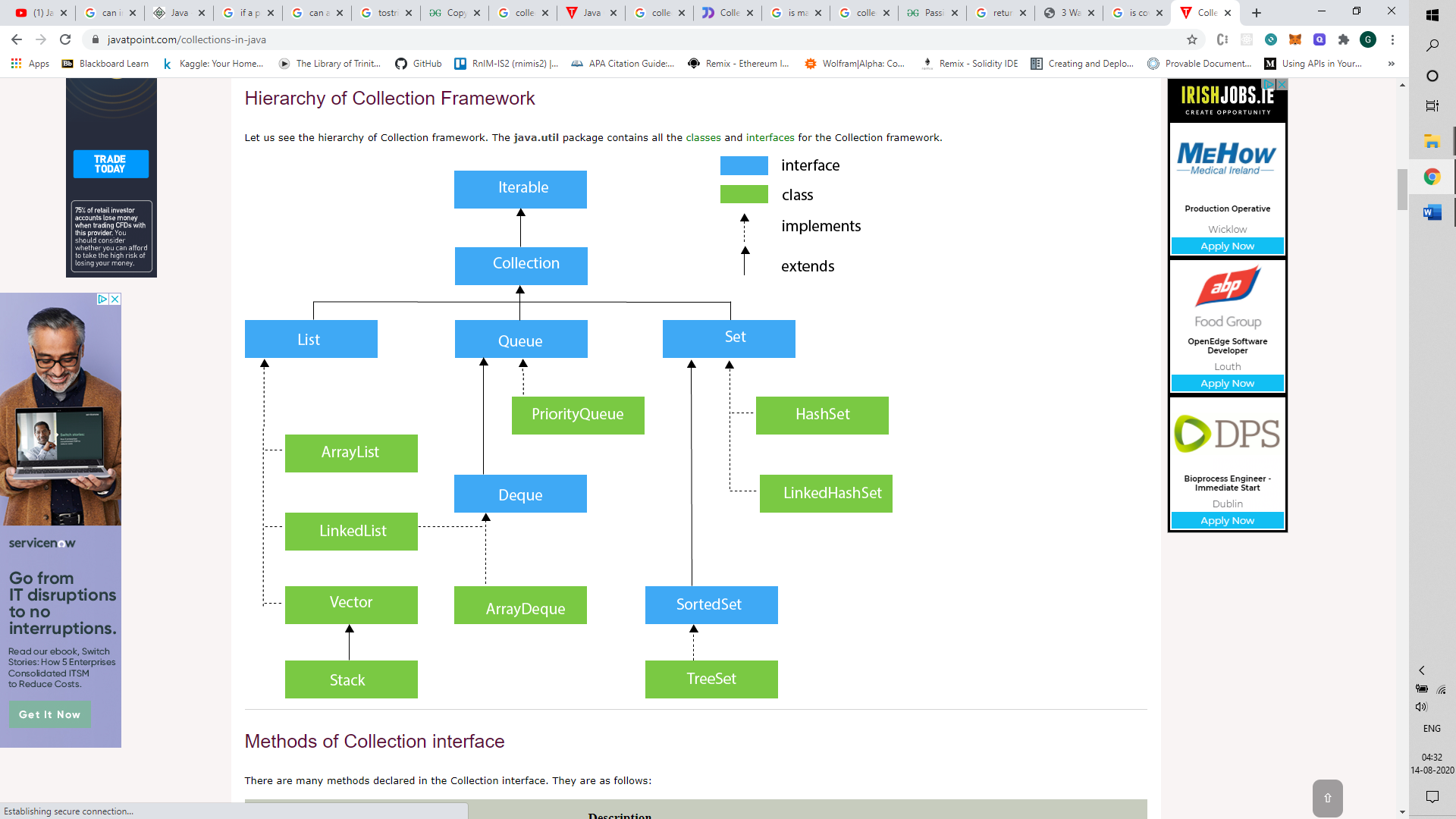
Collections.fill(list, “dummy data”);

Collections.frequency(list, “dummy data”);

Collections.max(fruits);

Collections.synchronizedList(fruits); // Also synchronizedSet, synchronizedMap

* Collection is an interface in Java. It is extended by List, Set, Map and Queue interfaces



**Nested Classes**

* Nested classes are classes in Java that are declared inside another class.
* They can further be classified into two – regular and static.
* Regular inner/nested classes can be instantiated only using an object of its outer class.
* Static inner classes can be instantiated without using an object of its outer class.

**HashMaps**

* It is a data structure in Java that stores data as key value pairs.
* A key is unique and is used to uniquely identify a value in a hashmap.
* Maps are not ordered.
* A key/value in a Map is case sensitive.
* E.g., Map<String, Integer> m = new HashMap<String, Integer>(4, 0.5f);

Here, Map is an interface implemented by the class HashMap. 4 is the initial capacity and 0.5f is the load factor. The 0.5f value indicates that the capacity of the map is doubled when the more than 50 percent of the map is utilized.

* Methods: put, replace, get, getOrDefault, compute, computeIfAbsent, computeIfPresent, putIfAbsent, remove, putAll(), entrySet().iterator().
* Difference between put and replace is that replace only works if the map contains the key mentioned in the replace() function.
* HashMap.keySet() returns a list of keys.
* getOrDefault() method is used to find a value for a given key and returns the default value passed as a parameter, if the key is not present.
* Remove() method is used to remove an entry with the specified key, passed as a parameter.
* Iterator<Entry<key,value>> iterator, iterator.hasNext(): used to iterate through a collection. hasNext() returns an Entry<key,value>.
* putAll() method is used to copy the map passed as a parameter into the function into another variable. This can also be done by passing the map as a parameter to the constructor during declaration.
* Map.forEach((key,value) -> anyCode;) is used to iterate through each key/value pair in the hashMap.
* Popular implementations of the Map interface are – HashMap, LinkedHashMap and TreeMap.
* HashMap – implementation: table/buckets; access speed O(1); no order
* TreeMap – implementation: red-black tree; access speed O(log(n)); natural order of keys
* LinkedHashMap – implementation: Table and linked list; access speed O(1); order of insertion (requires more memory than HashMap)

**Java Code Coverage and Analysis**

* Sonarqube – install sonarqube;
* Add 2 dependencies to pom.xml 🡪 jacoco and sonar-maven plugin
* Run project as maven build; goal: sonar:sonar
* View the report on <http://localhost:9000> 🡪 projects | issues

Using: C:\Program Files\sonarqube-8.4.1.35646\sonarqube-8.4.1.35646\bin\windows-x86-64>StartSonar.bat

**Log4j**

log = /log4j

log4j.rootLogger=DEBUG, FILE

# Define the file appender

log4j.appender.FILE=org.apache.log4j.FileAppender

log4j.appender.FILE.File=${log}/bookstore.txt

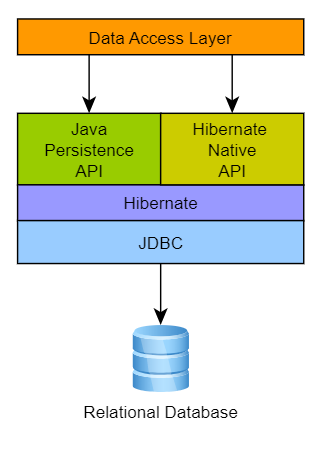
# Define the layout for file appender

log4j.appender.FILE.layout=org.apache.log4j.PatternLayout

log4j.appender.FILE.layout.conversionPattern=%-7p %d [%t] %c %x - %m%n

**Hibernate**

* ORM mapping tool that sits between Java data access layer and the relational database. It is a JPA provider.



* **Domain Model –**
* Types – value type and entity type. Value type refers to a column in a table, entity refers to a table.
* Naming strategies – explicit and implicit. Explicit uses annotations such as @Table(name=”users”) or @Column(name=”username”). Implicit annotations in hibernate is done by name the class name and the attributes of the class same as the name specified for the table and column names. This is implemented in hibernate using ImplicitNamingStrategy and PhysicalNamingStrategy.
* Hibernate uses the BasicTypeRegistry to map java types, hibernate types and sql types. It uses JDBC recommendations to map to the database.
* We can also implement custom basic types in two ways:
  + Implementing a basic type and registering it – see documentation
  + Implementing a user type which doesn’t require registration
* Enum mapping – ordinals and names.

E.g., public Enum PhoneType{

LandLine, Mobile;

}

@Enumerated(EnumType.ORDINAL)

@Column(name = “phone\_type”)

private PhoneType phoneType;

* @Nationalized annotation is used to make String java type be mapped to nclob data type rather than a varchar.
* We can either persist date or time using the following.

E.g.,

@Column(name = "`timestamp`")

@Temporal(TemporalType.DATE) / @Temporal(TemporalType.TIME)

private Date timestamp;

* @Id annotation is used to mark a field as a primary key.
* A primary key field can also me marked with an @GeneratedValue annotation.

E.g., @GeneratedValue(strategy = GenerationType.auto)

* Quoting is done to map fields that are keywords. Global quoting is done in the hibernate.cfg file.

E.g.,

@Column(name = "\"name\"")

private String name;

* @Generated annotation –

**Rest API**

They have the following classes: Model class, Service Class uses the repo class, Repo/Connector/DAO class, Repository

API controller, service testing using **Mockito**

Mockito is present within spring-boot-starter-test dependency (includes junit and Mockito)

@RunWith(SpringRunner.class)

@SpringBootTest

class BookstoreApplicationTests {

@Autowired

BookService service;

@MockBean

BookRepository repo;

@Test

void getAllBooksTest() throws ParseException {

List<Book> books = new ArrayList<>();

books.add(new Book(1, "Harry Potter and the Sorcerer Stone", "J K Rowling",

new Date(0)));

books.add(new Book(2, "Seventh Mountain", "Thomas Merton",

new Date(0)));

when(repo.findAll()).thenReturn(books);

assertEquals(2, service.getAllBooks().size());

}

}

Learn Access Modifiers, DataTable

Await and Async in java script

Lambda functions in Java

Serializable, ObjectInputStream, OutputStream, Synchronized

Git commands from eclispse, Mockito

Sessions in Java

Commands to identify and kill a process on port - netstat -ano | findstr :8081 | taskkill /PID <6140> /F

**Eclipse –**

* ctrl + shift + R to search for a file
* ctrl + shift + F to format a file