Web Full Stack

Java

Java is a platform independent programming language & object oriented programming language

Platform Independent: It can be run on any platform

Object Oriented Language: it helps to create real-world entities in the application where be these real world entities are called as objects which will have 2 main things

1. properties: What object has
2. behaviours: What object does

Main building blocks of any object oriented language is

1. class: Blueprint of an object that describes it
2. object: It is a real-world entity which is in stance created from the class, it will have the initialized values for their properties

Fundamentals of Java

1. Operators: +, -, \*, /, =, %, ++, --, <, >, ==, <=, >=, !=

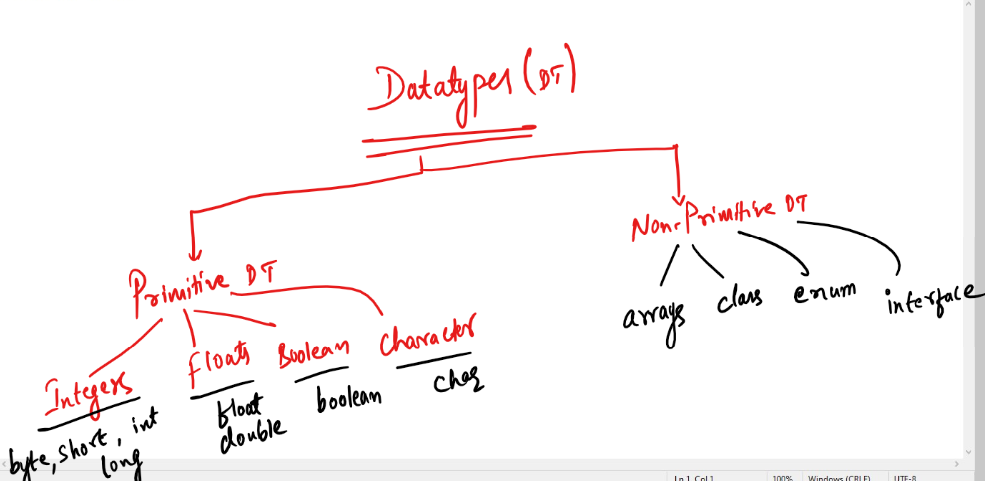


1. Datatypes: int, short, long, byte, double, float, boolean, char
2. Conditional Statements: if, else, switch
3. Loops: for, while, do-while
4. Arrays
5. Class & Objects

Java main entry point

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

Datatypes



Operators: These help to operate on data

Conditional Statements

When you want some statements that needs to be executed based on the conditions you can use conditional constructs like

* if
* if else
* if else if else if … [or] if else if else if ….. else
* switch

Syntax:

if (condition) {   
 statements  
}

else {  
  
}

Multiple conditional statements  
if (condition) {….} else if (condition) {……} else { …. }

switch (choices) { case choiceValue: statement; break; }

Loops

for loop: It is used when you know how many iterations you want to do

while loop: It is used when you want to execute statements repeatedly until condition is false

do while loop: It is like while, but executes the statements at-least and then checks the condition

Arrays: It is container that can store multiple values of same datatypes

String class: It is used to work with collection of characters, it provides methods like length(), concat(), toUpperCase(), toLowerCase(), charAt(), split() and so on.

Day 2 Agenda

* Methods
* Classes & Objects
* Constructors
* Keyword this
* OOPS Principles

Understanding method arguments & return types

public static void add(int x, int y) {   
 int result = x + y;  
 System.out.println(result);  
}  
public static String sayHi(String name) {   
 return “Hi “+name;  
}  
public static void main(String[] args) {  
 add(20, 30);  
 String result = sayHi(“Sachin”);  
}

Classes & Objects

Class is a template / blueprint of an object

Object is a real world entity which is an instance created from the class, it will have initialized properties & methods

How to call a constructor from another constructor

There are two ways depending on the scenario

1. When you want to call constructor of same class use ‘this(args)’ keyword
2. When you want to call super-class constructor then use ‘super(args)’ keyword

Note: call to this() or super() should always be the first line of the constructor

class A() {  
 A() {   
 ….  
 }  
 A(String a, int b) {   
 this(a); // A(String)  
 ……  
 }  
 A(String a) {   
 this(); // A()  
 ….  
 }  
}

Enhanced for loop

It simplifies iterating arrays and collections by providing a simple syntax

for(type t : collection) {   
  
}

int arr = {…}

for(int x : arr) {   
 instead of arr[i] you can use ‘x’  
}

User[] users = ….

for(User u : users) {   
 instead of users[i] you can use ‘u’  
}

Day 3 Agenda

* Static keyword
* Inheritance - extends, super
* Polymorphism - overloading & overriding
* final keyword
* Abstraction - abstract class & interface

Static keyword:

Static members are accessible without creating objects, you can use class-name to access static members.

Static members are loaded before object is created which means at the time class loading

Inheritance:

Process of acquiring properties & behaviours of a class from another class, you use extends to achieve inheritance, then use super() to call the constructor of parent class from sub-class constructor, you can also use super(args) when you want to call argument constructor of the parent class

Note: super() must always be in the first line of the constructor

Note: this() can also be in the first line, but when you use this() or this(args) it calls constructor of the same class, however you can’t keep both this() or super() in the first line so you can keep either of the one, when you use this()/this(args) then super() will be ignored by compiler

Polymorphism

A method with many forms

* Overloading
* Overriding

Accessing subclass members from superclass reference

Person p1;  
p1 = new Person(…);  
p1 = new Employee(….);  
p1 = new Student(….)

If you have super-class reference then you can only access members present in superclass but not the members present in sub-class

p1.setName(..); // valid  
p1.setGender(..); // valid  
p1.display(); // valid  
p1.setId(..); // CTE  
p1.setSalary(..); // CTE  
p1.setGrade(..); // CTE  
p1.setRollNo(..); // CTE

Abstraction:

Hiding the complexity and showing only the necessary details to the end user so that the user could use the object easily

Abstraction divides the roles & responsibility of the end users so that they can concentrate on their work without thinking of others work, this makes code flexible to the changes so that when a code is modified at any one’s end then others code need not to be modified.

Note: Abstraction is achieved with method declarations or abstract methods or methods without body

// abstract method

abstract void m1();

// non-abstract method

void m2() {   
  
}

Abstraction is achieved in two ways

1. interface - 100% abstraction - all the methods are abstract by default
2. abstract class - partial abstraction - not all the methods are abstract some methods have body and some don’t.

Day 4 Agenda

* Interfaces
* Abstract classes & Methods
* Access Modifiers
* Layered Architecture

Layered architecture:

It is used to separate presentation logics, navigation logics & model logics in the application so that it can be easily maintained in the large scale applications, each layers are completely abstract with other layers so that any changes done in one layer doesn’t force to change other layers.

Layered architectures makes your code more reusable especially Model layer and also you must have a loosely coupled code, if you make changes in one place then the dependent codes need not to modify

ex: Controller depends on Model Layer object

There 3 different scenarios that controller can use to use the model layer object

Scenario1

Using model layer class as a reference & creating the object, it makes code tightly couple because if the model layer class changes then controller code should also be changed, if the controller is doing the same task then it doesn’t need to change its code if the implementations for such tasks are changing.

Controller Code

OracleDb db = new OralceDb();  
db.save(); db.find(); db.modify()

If the new implementation to perform same tasks needs to be used then you need to change the controller code and also the new implementation is not forced to implement same methods of OracleDb, hence we need to have an interface

Scenario2

interface DB { save(); find(); modify(); }

This interface can be implemented by any number of classes

class OracleDB implements DB { /\* implement all methods \*/ }

class MySqlDB implements DB { /\* implement all methods \*/ }

Controller Code

DB db = new OracleDB();   
db.save(); db.find(); db.modify()

In this scenario controller code is still modified if the new implementation object needs to be used i..e, DB db = new MySqlDB();

Scenario3

Abstract object creation of these dependencies using some design pattern like Factory Design pattern.

Factory Design pattern: This helps to returns the object by abstracting the object creation at the caller side

Implementing layered architecture with User, UserDao, ObjectFactory, ViewController, UserDaoArrayImpl

We need to create following list of Java files

1. User.java >> com.hsbc.model.beans
2. UserDao.java >> com.hsbc.model.dao
3. UserDaoArrayImpl.java >> com.hsbc.model.dao
4. ObjectFactory.java >> com.hsbc.utility
5. ViewController.java >> com.hsbc.ui

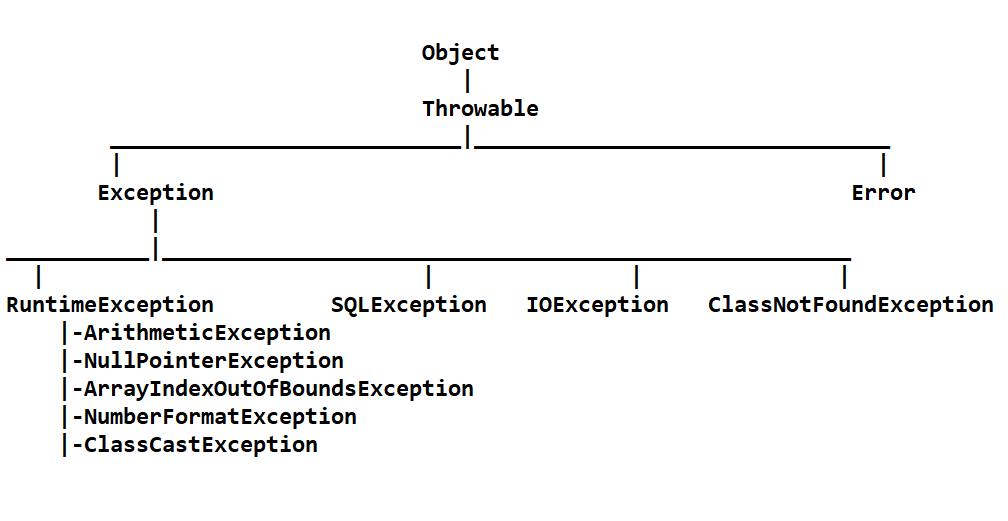
Exception Handling

Exceptions are runtime errors which abruptly terminate the application if in case not handled

There are 5 keywords in Exception Handling mechanism

1. try
2. catch
3. finally
4. throw
5. throws

Java has provided some predefined exceptions



Object class methods

1. toString(): Returns the object properties, but by default returns memory address
2. equals(Object): Returns boolean while comparing two objects, by default compare two objects’ address
3. hashCode(): Returns the hashCode of the object, by default returns object address in int

How you can override them

1. toString(): override it to return properties so that when any code prints the object it calls toString() automatically
2. equals(): Override it to compare two objects properties to identify duplicates ex: two employees id you can compare or two customers id you can compare
3. hashCode(): Override it to return an unique id for each object, so that it can be searched using that unique id

Note: Both equals() & hashCode() must be overridden together, if you want to represent uniqueness of any object when you store it in the Set or Map (Will be discussed in Collection Framework).

Multithreading

A program that can perform more than one task simultaneously .

In Java you can make more than methods to run simultaneously

Day 7

Collection Framework API’s

List

Set

Map

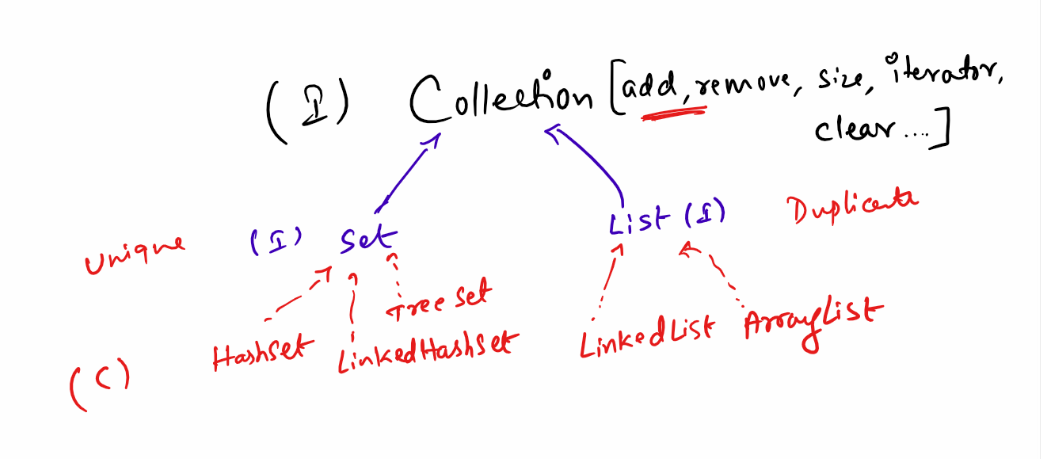
Comparable

Comparator

SQL

Collection Framework

Collection is a container which can store multiple objects and it is dynamic in nature, it provides many inbuilt methods to main the objects, it provides various implementations where you can have only unique elements, duplicate elements, sorted elements, elements in sequential order, elements in random order and so on



Collection is a root interface with methods like add(), remove(), size(), iterator(), clear() and so on, it has 2 sub-interfaces

1. List: Supports duplicates
2. Set: Supports uniqueness

List: It will have an implementations that maintains elements in sequential order, how you store the same order the elements are maintained, it supports duplicates

List has 2 implementations:

1. LinkedList: Stores elements in non-contiguous memory address, adding/removing the elements are faster compare to iteration, because it uses link between one element to another and each elements might be stored in non-contiguous memory address
2. ArrayList: Stores elements in contiguous memory address, adding/removing will shift the elements to different memory address to maintain contiguous memory address, retrieval is faster in array list compare to adding/removing elements

Set: It will have implementations that will support only unique elements, it has 3 implementations

1. HashSet: Retrieval is faster
2. LinkedHashSet: Maintains elements in insertion order
3. TreeSet: Maintains elements in sorted order

How Set identify duplicates

Set internally uses equals() & hashCode() method to identify the duplicates, these two methods are present in Object class.

equals(Object ob): It compares two objects

hashCode(): It returns the hashCode of the object that must be unique for every object, it is useful at the time of adding/removing/searching

Note: Primitive types equals() and hashCode() are internally used by its value, however for complex types we must override equals & hashCode

All the collection API’s use generics to mention the type of data it wants maintain

ex: If you want to maintain String then you can use Set<String>, if you want to maintain only int, then you must use Set<Integer> not Set<int> this is wrong, Integer is a class in Java that represents int type, if you want to maintain double then use List<Double> not List<double>, because Double is the class that represents double type, similarly we have corresponding wrappers classes for every primitives

Set<Customer> it maintains customer objects

List<Employee> it maintains employee objects

Comparable<T>: It is used by inbuilt classes who wants to sort the objects, it has a method called compareTo() which returns int value, if the value is +ve then it means the object is bigger, if the value is -ve then object is smaller, if value is 0 it means its neutral.

But Comparable<T> can’t sort multiple properties it can only sort one property at a time, if you want to have multiple sorting logics for different properties then you must use Comparator<T> it has a method compare(Object, Object).

Comparator<T> is an interface with one method compare(Object, Object), this needs to be implemented separately and use the object of the Comparator instance in the sorting class i.e., TreeSet.

TreeSet automatically sorts if the object implements Comparable, however you can use Comparator and make treeset to use different sorting technique by passing comparatorImplemenation as a argument.

How to sort List<T>

You can use a class called Collections that has a static method ‘sort(List, Comparator)’ Collections.sort(List<T>, comparatorImplementation)

Example

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

list.add(…….); // assume you have some employee objects then you can sort it using

Collections.sort(list, comparatorInstance);

Map:

It helps you to maintain data in key value pairs, it is not part of the collection but works similar to collection.

Key must be unique, but value can have duplicates, Map has methods like

put(key, value): To store  
get(key): To retrieve   
remove(key): To remove  
keyset(): Returns all the keys in Set format

Map is an interface with 3 implementations

1. HashMap: Random order
2. TreeMap: Sorted order
3. LinkedHashMap: Insertion order