Topics

* Core Java
* JDBC, Servlets & JSP
* HTML, CSS, Typescript, Javascript
* Git
* UML
* Junit
* Spring Framework
* Spring Boot & Microservices
* Python
* Cloud & Containers
* Design Patterns
* React.js

Java

It is a platform independent & object oriented programming language.

Platform Independent: You can run java programs on various platforms without re-compiling

Object Oriented Language: You create real world entities in the application and define the properties & behaviors of these real world entities, these real world entities are called as objects.

Two things are must in object oriented language which are also called as building blocks of an object oriented language

1. class: blueprint or template of an object
2. object: an instance created from the class

run -> entry point -> C/C++/Java -> main method

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

class Hello {   
 public static void main(String[] args) {   
 System.out.println(….);  
 }  
}

String[] - it is a command line argument which accepts the input from the user before launching the main

static - you can access static members without creating the object

public - visible outside the class

Softwares required

1. JDK - javac, java commands
2. Editor - notepad, vscode or IDE like Eclipse

Eclipse -> File -> Other -> Java Project

Fundamentals

1. Variables - these are the memory to store the data
2. Datatypes - byte, short, int, long, float, double, char, boolean, arrays & complex types
3. Operators - \*, -, +, /, ++, --, <=, >=, ==, >, <, %, !=, &&, ||
4. Conditional statements - if else .. if else if .. else, switch
5. Loops - for, while, do while
6. Jump statements - break, continue, return

Java Naming conventions

avoid using variable names like a, b, c, i, j, k, x, y, z as they don’t explain themselves what they are instead use variable names like name, age, phoneNumber, gender and so on

Similarly for methods / functions you must use the names which are having meanings like

display(), update(), getDetails(), searchEmployee(), searchCustomer() and so on, avoid using test(), demo(), abc() and so on.

While or Do While loop

While loop: It first checks the condition & then executes the loop

Do while loop: at least once the loop is executed and then the condition is checked to continue

for each or enhanced for loop iterates the elements without using the index.

Activity:

Create an array of numbers like 1, 4, 3, 6, 2, 5, 0, -5, iterate over an array and display the maximum, minimum and sum of elements in the array  
ex: Max: 6, Min: -5, Sum : 16

Note: Use only one loop to display max, min & sum

For the same array search an element 2 that must print 2 is found  
then search an element 9 that must print 9 is not found

Command line arguments: These are the inputs which you can provide while running the program so that programs can use these inputs before starting, the String[] in the main is a command line argument you can pass while using java command or in eclipse you can pass through Run As option.

i.e.,

java TestApp 20 30 40   
java TestApp Hello Everyone

Type conversion: Converting one type to another type, it can be automatic or explicit depending on the type of variable you are using

Auto-widening (Type promotion)  
int x = 25; // int = 4, long = 8  
long y = x; // type promotion

Explicit-narrowing

long x = 25; // long = 8, int = 4  
int y = (int)x; // explicit narrowing

Auto-boxing & Auto-unboxing: These are automatic because it deals with converting primitives to wrappers and vice versa

int x = 25; //  
Integer y = x; // Integer = int >> Auto-boxing, 25 will stored as an Integer object

int z = y; // int = Integer >> Auto-unboxing

class Calculator {   
void add(byte x, byte y) {   
  
 }  
}  
int a = 25;  
int b = 30;  
add((byte)a, (byte)b);

Object Oriented Programming

class: It defines the structure, it can have following members

* variables or fields
* methods: Will have reusable logics
* constructors: are called when objects are created, you can have initialization logics

object: It is an instance of a class which is created with new keyword

User user1 = new User();

Constructor vs Methods

|  |  |
| --- | --- |
| Constructor | Method |
| Constructors are called when objects are created | Methods are not called when objects are created |
| It will not have return type | It must have return type |
| Its name must be same as the class name | It’s name need not to be same as the class name, however you can give class name but they are invoked only when you call them explicitly |
| It usually will have initialization logic | It will have reusable logics |

Activity:

Create User class with id, name, constructor to initialize the id & name, display method to print id & name, in main method create 2 objects & call the display method.

Classes can have static and non-static(instance) members

static members you can access directly from the class name, however non-static members you can access only by creating the object whenever you want these members to accessed outside the class.

class User {   
 int id;

static int counter = 5;  
}  
User.counter = 7; // valid  
User.id = 33; // invalid, because id is instance variable

static members are loaded at the time class loading, non-static members are loaded at the time of object creation.



Quiz:

int x = 20;

int y = ++x;

System.out.println(y); //21  
System.out.println(x); // 21

1. class A {   
    static void demo() { test(); } // error  
    void test() { System.out.println(“hello”); }  
   }
2. class A {  
    static void demo() { A a = new A(); a.test(); } // ok  
    void test() { System.out.println(“hello”); }  
   }

}

class A {   
 // default constructor in case you have not written  
}   
class B {   
 B(int x) {   
 }  
 // only one constructor will be created which takes int parameter  
}

byte x = 30;  
long y = 40;  
int z = (int) y; //   
int z1 = x; //

Principles of OOPs

Inheritance  
Encapsulation  
Polymorphism  
Abstraction

Inheritance:

Acquiring the properties & behaviors of a class from another class, you use extends keyword to inherit   
class Person {   
 name, gender, email, dob // properties  
 display() { } // display data   
}  
class Employee extends Person {   
 employeeId, salary   
}  
class Student extends Person {   
 rollNo, grade  
}

Note:

* In Java you can achieve various types of inheritance like single level, multi-level, hierarchical, multiple(supported only through interface) & hybrid (supported only through interface)
* If a class doesn’t extend any class it automatically extends Object class
* Subclass constructors calls the default constructors of the parent class by default, however you can pass parameters to the super() to call the parameterized constructor of the parent class

Polymorphism:

A method that has many forms or a method that can do more than one job, there are two types of polymorphism

1. Compile time – overloading – method invocation is determined at compile
2. Runtime – overriding – method invocation is determined at runtime

class Person, class Employee extends Person, class Student extends Person { }   
  
Method Overriding: you will create the same method of super class in the subclass with different logic & to achieve runtime polymorphism you will use the super class reference variable and call the overridden method  
void test(long x) { }   
int a = 20; test(a);  
long b = 30; test(b);  
byte c = 40; test(c);

Method Overloading: You will create multiple methods with the same name in the same class with different signature(parameter type or order or return type)

Encapsulation: It is mechanism of hiding the data by making data private & the only way you can access the data is using public methods, private data will not be visible outside the class so that it restricts the direct access from the outsider

class Employee {   
 private int id;  
 private int age;  
 Employee(int id) { this.id = id; }  
 public void setAge(int age) { this.age = age; }  
 public int getAge() { return age; }  
 public int getId() { return id; }  
}  
Employee emp = new Employee(34);  
emp.id = 356; // error  
emp.age = 4939; // error

Abstraction:

Hides the complexity and shows only the necessary details so that it will be easy to use the object, here the developers need not to worry about how the method is implemented, instead they must know what a method does.

We achieve abstraction by creating abstract methods in 2 ways

abstract methods: These are the methods which doesn’t have logic or body

1. interface: Will have only abstract methods & constants
2. abstract class: can have both abstract methods (methods without body) & non-abstract methods (or methods with body)

interface: Methods are by default abstract & public, variables are by default public, static & final, however you can’t have constructors, hence you can’t create the object of the interface

interface UserDB {   
 void store(); // public abstract void store();  
 void delete();  
}

A class can implement the interface (a class can also implement one or more interfaces)

D4 implements the interface  
class UserDBOneImpl implements UserDB {   
 // mandatorily you must implement all the methods of interface  
}

D1 ui1(UserDB db) { db.store(); }  
D2 ui2(UserDB db) { db.delete(): }  
D3 ui3(UserDB db) { db.store(); }

D5 ui1(new UserDBOneImpl());

Customer -> login(), transfer(), getDetails(),

Employee -> deleteAccount()  
  
interface CustomerOP {   
 login(); transfer(); getDetails();  
}  
interface EmployeeOP {   
 deleteAccount();   
}  
class BankOp implements CustomerOP, EmployeeOP { 4 methods }  
Customer -> ui(CustomerOP cop) { cop.login(), cop.getDetails(), cop.transfer() }  
Employee -> ui(EmployeeOP eop) { eop.deleteAccount(); }

Abstract class

It allows you to have abstract methods and methods with body, go for abstract class when you don’t know the full implementation of the class

ex:

interface DataStructure {   
 void add(int x); // LIFO, FIFO, SO, RO, Uniqu  
 void delete(); // LIFO, FIFO, SO,   
 void findAll();  
}  
abstract class AbstractDS implements DataStructure {   
 void findAll() { … }  
}   
class StackDS extends AbstractDS {   
 // mandatorily implement all the abstract methods like add, delete  
}   
class UniqueDS extends AbstractDS {   
 // mandatorily implement all the abstract methods like add, delete  
}  
Note: Abstract class can have constructors, but you can’t create object of an abstract class

Note: From Java 8 there are few changes done in the interface

Difference between interface and abstract class

|  |  |
| --- | --- |
| Interface | Abstract class |
| All the methods are by default abstract & public | You need to explicitly use abstract keyword |
| All the fields are public, static & final | Fields are not constants unless explicitly defined |
| Can’t have constructors | Can have constructors |
| Provides full abstraction | Provides partial abstract |
| An interface can extend one or more interfaces which is called as multiple inheritance | An abstract class can extend only one class |
| We can’t instantiate | We can’t instantiate |
| More secured when you want to share the interface | Not secured when you want to share the abstract class to other developers |

Packages & Visibility of members

For visibility java has 4 access specifiers

class A {   
 private int x; //   
 int y; // default or package private  
 protected int z;   
 public int i;   
}

1. private: Visible only within the class
2. no keyword i.e., default or package private: Visible within the package
3. protected: within the package & outside the package only to the subclasses
4. public: Visible to all

package com.ibm package com.org;  
public class A { class B {   
 protected int z; // can’t access z  
} class C extends A { // z will be inherited

Exception Handling

5 keywords – try, catch, finally, throw & throws

try { }: try block should have code that might generate exception, ex: accessing the DB, accessing the files, working with remote service

catch(..) { } : catch block will handle the exceptions generated in try block, you can have multiple catch blocks

finally { } : This is absolutely executed whether or not exception handled, some mandatory statements that needs to be called like closing the files, db connections

throw: It is to manually generate an exception or user defined exception, ex: UserNotFoundException, AgeInvalidException, AccountNotFoundException, InsufficientBalanceException

throws: It is to forward the exception from one method to another when a method wants a caller to handle the exception, because caller knows what to do with the exception.

Exception hierarchy



Error you can’t handle, but Exceptions you can handle, here there are 2 types of exceptions

1. Checked exceptions, which must be handled at the compilation time using try – catch or it must be propagated using throws
2. Unchecked exceptions are subclasses of RuntimeException which are not forced to handle at the compilation time, because these exceptions are programmers mistakes that can be resolved with the proper logic

try with resource:

It is introduced in Java 7, to automatically close the resources like Files, Databases, Scanner to reduce too much try catch finally, this syntax you can use only on the classes that implement AutoCloseable or Closeable

try ( Scanner scan = new Scanner(System.in);   
FileReader reader = new FileReader(scan.next()); BufferedReader buffer = new BufferedReader(reader) )   
{  
  
}

try( String str = “hello”; ) { }// this gives compilation error

Custom exceptions

Creating exceptions of our own based on the project requirement by extending any one of the Exception class, you can extend any one of the exception class like Exception or RuntimeException, to create checked exception you can extend Exception or sub class of Exception except RuntimeException, if you extend RuntimeException it becomes unchecked.

ex: InsufficientBalanceException, AgeInvalidException, LimitExceedException and so on.

class EmployeeNotFoundException extends Exception {   
 // minimum 2 or more constructors you must create  
}

Employee find(int id) throws EmployeeNotFoundException {   
 if(…) { throw new EmployeeNotFoundException(…); }  
}

Assertions: These are used to test some results in your code, if the result is not as per the expectation then you can make assert false that can throw an assertion error, it is used only at the time of development / testing

int age = scan.nextInt();

assert (age >= 18) : “Age is less than 18”;

This works only if you enable the assertion, you get an assertion error if the assert condition is false.

By default assertions are disabled, to enable you must use -ea option in the vm argument i.e., java -ea in case you are using command prompt, else in Eclipse you need to use configuration in Run As option

Multithreading

Running more than one task at the same time, means at the same time multiple threads can run the methods by sharing the CPU time

There are mainly two ways to write multi threaded applications in Java

1. By extending the Thread class
2. Another is by implementing a Runnable interface, it has run() method that is an entry point for the threads

Note: Thread class implements Runnable interface internally

To create threads you must use Thread class that provides various methods to manage threads

Thread thread1 = new Thread(runnableTypeObject);  
Thread thread2 = new Thread(runnableTypeObject)

thread1.start(); // it registers the threads in the thread schedular

thread2.start(); // 2nd thread registers in the thread schedular

Once the threads are registered run() method logics will be executed

Thread.currentThread(): returns the currently running thread  
Thread.sleep(long): thread goes to sleep state

getName(): returns the name of the thread

setName(String): to provide the name for the thread

Executor Framework

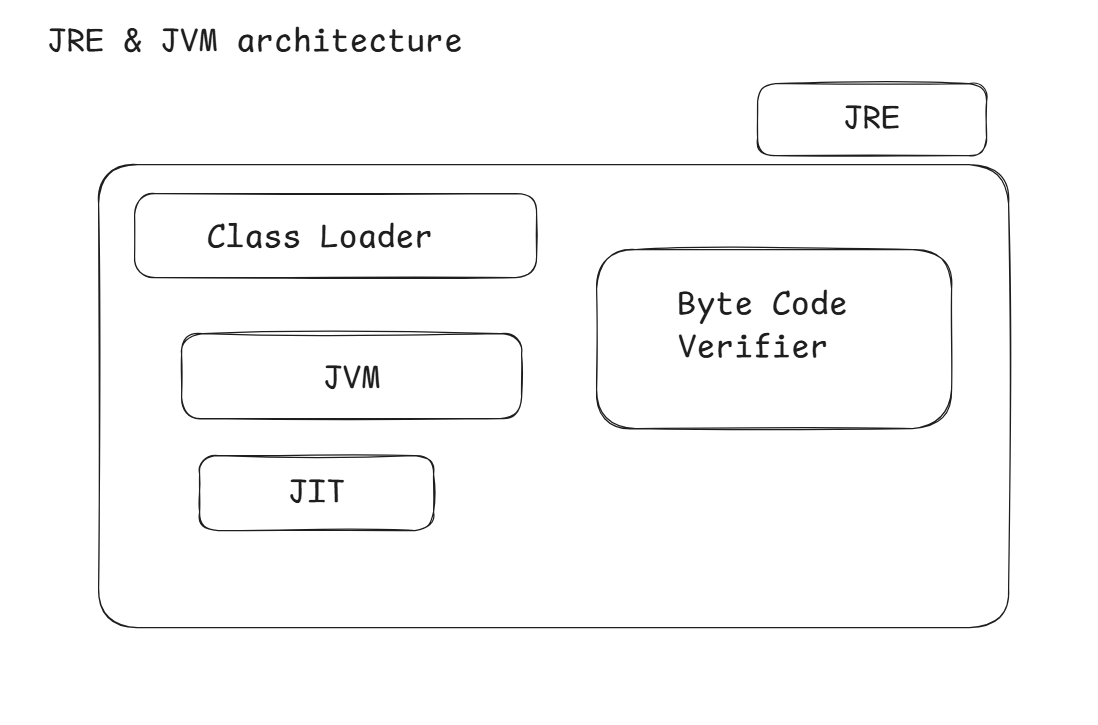
it takes care of creating the threads and allows to reuse the threads in the thread pool.

Thread t = new Thread(..); // here thread will be created & destroyed once the job is done

Executor executor = Executors.newFixedThreadPool(3);

executor.execute(runnableType);

JRE & JVM architecture & Garbage collector



Garbage collector is a thread, which takes care of marking all the objects that has no reference & remove those objects automatically so that it gives space for other objects

main() {   
 test();  
 demo();  
}  
test() {   
 A a = new A();  
 B b = new B();  
}  
demo() {   
 X x = new X();  
}

Here once the test() method is completed its execution, a & b references will be removed whose objects are eligible for garbage collection, when garbage collector marks the object it will find two types of objects.

1. object with reference
2. object without reference – only these objects will be removed.

String class

It creates an immutable string object, once string object is created you can’t modify its content.

String methods:

concat(), toUpperCase(), toLowerCase(), charAt(), length(), equals(), equalsIgnoreCase(), substring(), indexOf(), split(), strip(), isBlank() and so on.

String s1, s2, s3, s4, s5;  
s1 = “hello”;  
s2 = “hello”; // s1 == s2 >> true   
s3 = new String(“hello”); // s1 == s3 >> false  
s4 = new String(“hello”); // s3 == s4 >> false

s1.equals(s3); // compares content instead of address  
s1.concat(“123”);  
System.out.println(s1); // hello  
s1 = s1.concat(“12345”);  
System.out.println(s1); // hello123

Inner class

These are the classes that you can nest inside another class, you use nested classes when outer class is dependent on another class:

ex: Thread class is having a nested enum called State, enum is also a type of class but will have fixed set of constants.

There are four types of inner class you can create

1. Static inner class
2. Non-Static inner class
3. Anonymous class
4. Local inner class

enum: it is a type of class which helps you to create fixed set of named constants

enum Gender {   
 MALE, FEMALE  
}  
You can create a variable of Gender which can accept either MALE or FEMALE value onlye  
Gender gen1 = Gender.MALE;  
Gender gen2 = Gender.FEMALE;  
gen1 = Gender.XYZ; // error

Instead of create String gender you can create an enum Gender

Event Management System Use Case

class Event {   
 id, name, time, location, organizerNames, startDate, endDate; // properties  
 updateEvent(startDate, endDate) { }   
 updateEventTime(time) { }   
 display(id) { … } // display the event details based on the id  
}  
main method -> you can create 3 to 4 event objects -> display all the events -> updateEvent() & display the updatedEvent

Day 4 topics

1. Collection with Generics
2. IO Streams – File handling & Serialization(optional)
3. Reflection API
4. Java 8 features
5. Java 11 features
6. Java 17 features
7. HttpClient
8. Internationalization
9. Arrays
10. Object class methods
11. Java annotations - @Override, @SuppressWarning
12. Debugging
13. Interview questions

Collection Framework

Collection can maintain multiple objects and it is dynamic in nature, it has predefined classes & interfaces that can maintain the objects in various forms

Arrays are also a collection but its size is fixed & you need to write the algorithm to maintain the elements in various forms

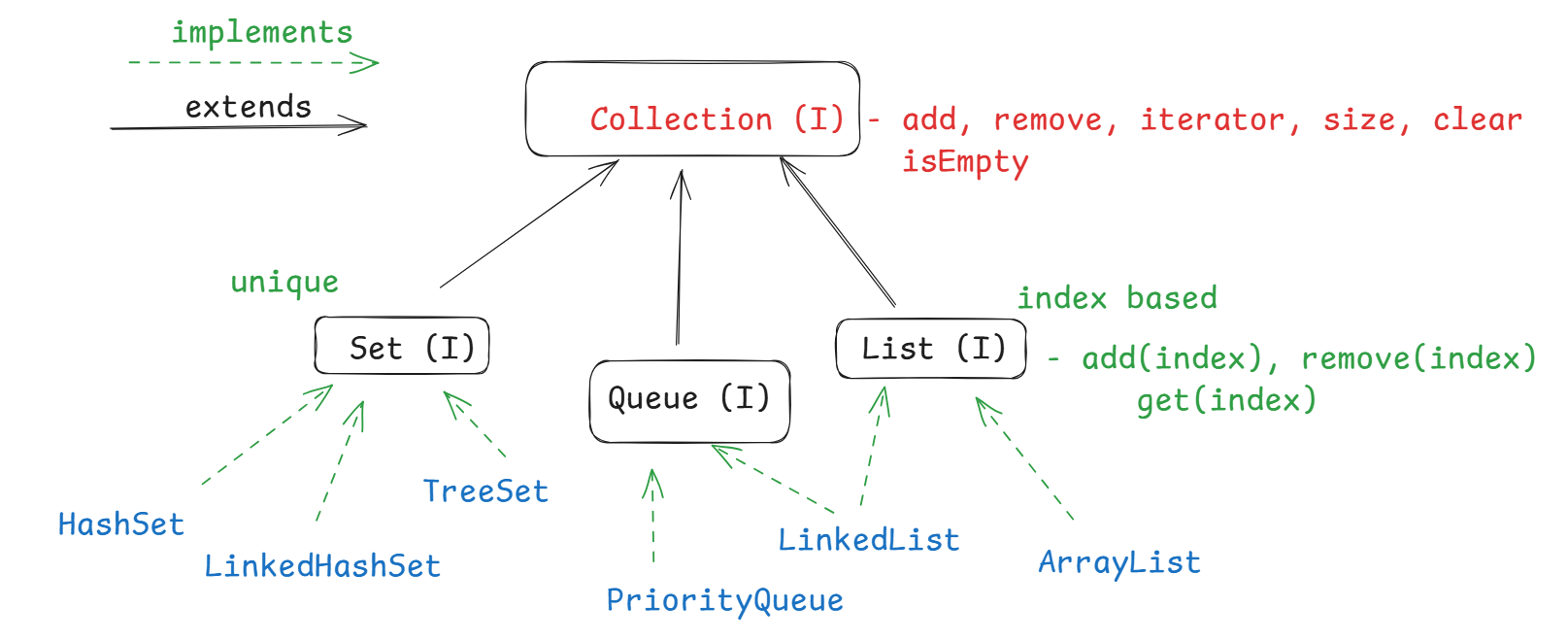
Employee[] employees = new Employee[5]; // object array

// employees can store maximum 5 employee objects, from employees[0] to employees[4]

Collection vs Array

|  |  |
| --- | --- |
| Arrays | Collections |
| Size is fixed | Size is dynamic |
| Manually you need to write algorithms to maintain elements in various forms like unique elements, sequential elements, sorting elements and so on | You get inbuilt methods to add, remove, iterate the elements |

Collection framework provides various interfaces & classes, below is the collection hierarchy



Note: You don’t need to know how these algorithm is internally implemented, instead you must know how they maintain the data

Set: Supports only unique elements  
TreeSet: Maintains the elements in sorted order, ex: storing student based on result  
LinkedHashSet: Maintains the elements in Sequential order, ex: online token based processing

HashSet: Maintains the elements in Random order, but it is faster in retrieving the data, because it uses hash based algorithm

Queue: It is used to process the data after removing, it provides methods like poll() to remove the element

PriorityQueue: It removes the element based on the priority i.e., sorted order

ex: online bidding

stock market [ order1(1000), order2(1005), order3(990) ]

LinkedList: It removes the elements in FIFO order  
ex: messaging apps

poll in Queue vs remove in Collection: poll() is part of Queue that can remove the element either in sorted order or FIFO order, however remove in Collection removes the particular element

List: It is index based

ArrayList: Maintains the elements in contiguous memory address and if the elements are added in between then other elements are shifted to different memory to maintain the contiguous memory address

ex: Use ArrayList in the application where you frequently retrieve and rarely add or remove the elements, like university applications

ArrayList adding and removing is slower & retrieval is faster

LinkedList: Maintains the elements in non-contiguous memory address, adding & removing is faster & retrieving is slower

ex: Online shopping apps

Collection can store multiple objects however storing multiple objects would be problematic at the time of retrieval hence collection uses generics to define what type of objects it will store

Without generics

List al = new ArrayList();   
al.add(5);  
al.add(88.5);  
al.add(true);  
al.add(new Employee(…));  
al.add(“hello”);  
With generics// it was introduced in Java 5  
List<Integer> al = new ArrayList<Integer>();  
al.add(5);  
al.add(8);  
al.add(“hello”); // compilation error  
al.add(44.5); // compilation error

Note: You must use only classes in generics you can’t use primitive data types

List<int> al = new ArrayList<int>(); // wrong  
List<Double> al = new ArrayList<Double>(); // correct  
List<Employee> al = new ArrayList<Employee>(); // correct

Iterator: It is an interface used to iterate the collection, it has 3 methods

1. hasNext: checks next element is present if present returns true else false
2. next: returns the next element
3. remove: removes the iterated element

Collection has iterator() method which you must call to get the Iterator reference  
Iterator<Integer> itr = al.iterator();  
while(itr.hasNext()) {   
 int ele = itr.next();  
 if(ele == 5) { itr.remove(); // removes the element 5 }  
}

Example  
List<Integer> al = new ArrayList<Integer>();

al.add(50); al.add(20); al.add(30); al.add(5); al.add(40); al.add(5); al.add(20); al.add(5);

al [50, 20, 30, 5, 40, 5, 20, 5]

al.remove(5); // remove the element present in 5th index

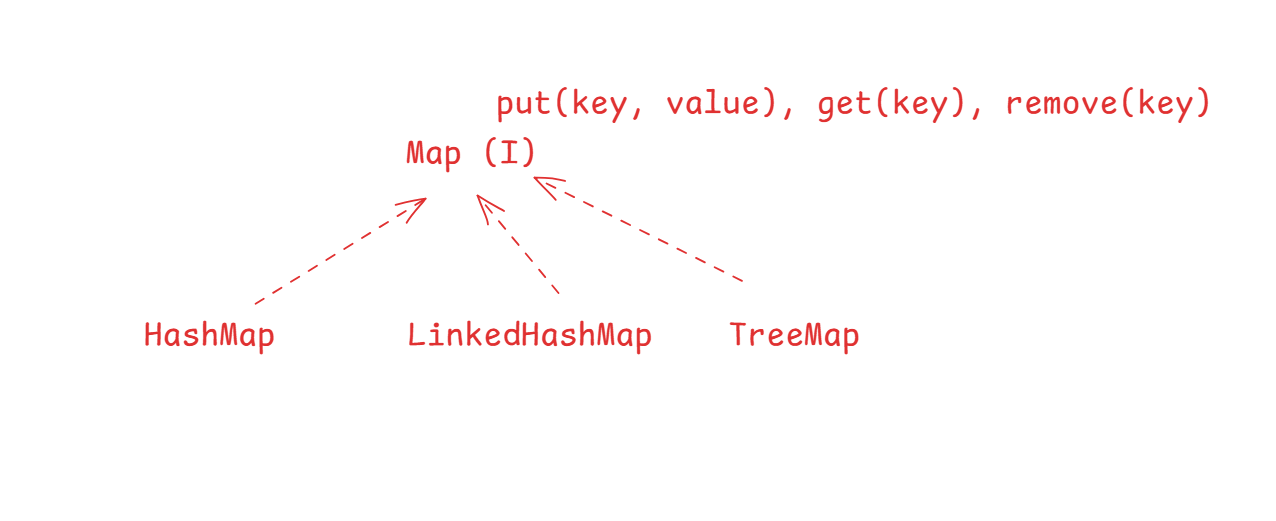
al.remove(Integer.valueOf(5)); // removes the value 5

Iterator<Integer> itr = al.iterator();

v = 5

itr.hasNext() -> it.next() == 5 -> itr.remove()

Map: It is also a datastructure where it maintains the data in key & value pairs, it is not part of collection hierarchy but it is also dynamic in nature.

Map<Key, Value>: it doesn’t support duplicates, key must be unique, value can have duplicate, it has methods like

* put: stores the data
* get: to read the data based on key
* remove: to remove the data based on key

HashMap: maintains random order

LinkedHasMap: maintains sequential order

TreeMap: maintains the key in sorted order

Note: Map & their implementations are present in java.util package.

Map<Integer, String> map = new HashMap<Integer, String>();  
map.put(123, “Raj”); map.put(456, “Vijay”); map.put(345, “Arun”);  
S.o.p(map.get(123));  
S.o.p(map)  
map.remove(123);  
S.o.p(map);   
change HashMap to TreeMap & LinkedHashMap & observe the output

You can maintains database credentials or server informations in the map in key & value structure

map.put(“dbuser”, “admin”);  
map.put(“dbpwd”, “Welcome@123”);

Java 8 features

1. Functional interface or Functional programming
2. Lambda expression
3. New Date & Time classes
4. Default & Static methods in interface

Activity

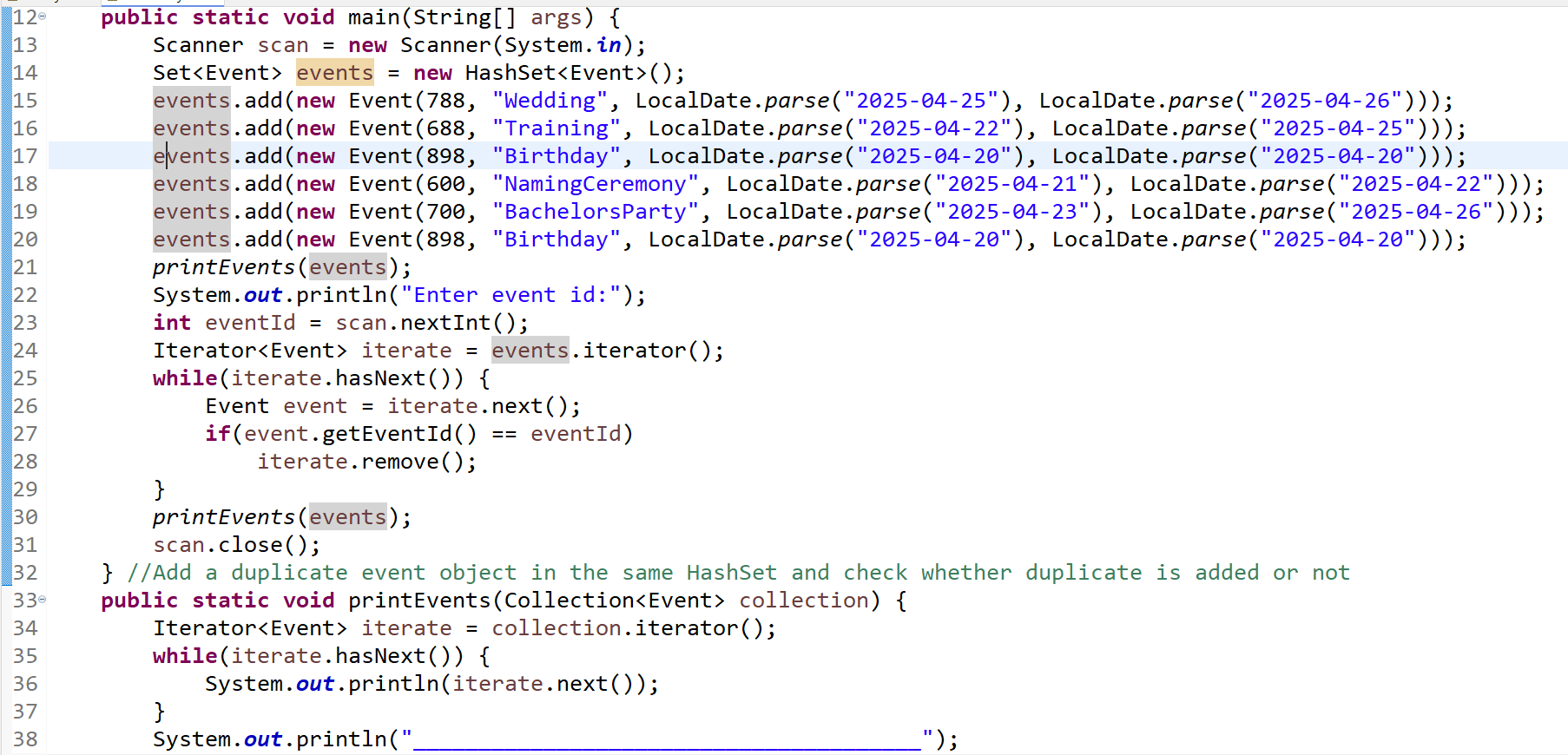
Create a class Event with following properties like eventId, eventName, startDate, endDate, constructor to initialize all the four properties, toString method to return all the properties, getters, setters for all the properties, in a separate class create the main method add 5 Event objects to the HashSet and print each event object by iterating the HashSet, next ask user to enter an event id and remove the event object from the HashSet which matches to the eventId, then again print the event objects present in the HashSet.

Add a duplicate event object in the same HashSet and check whether duplicate is added or not

Event.java

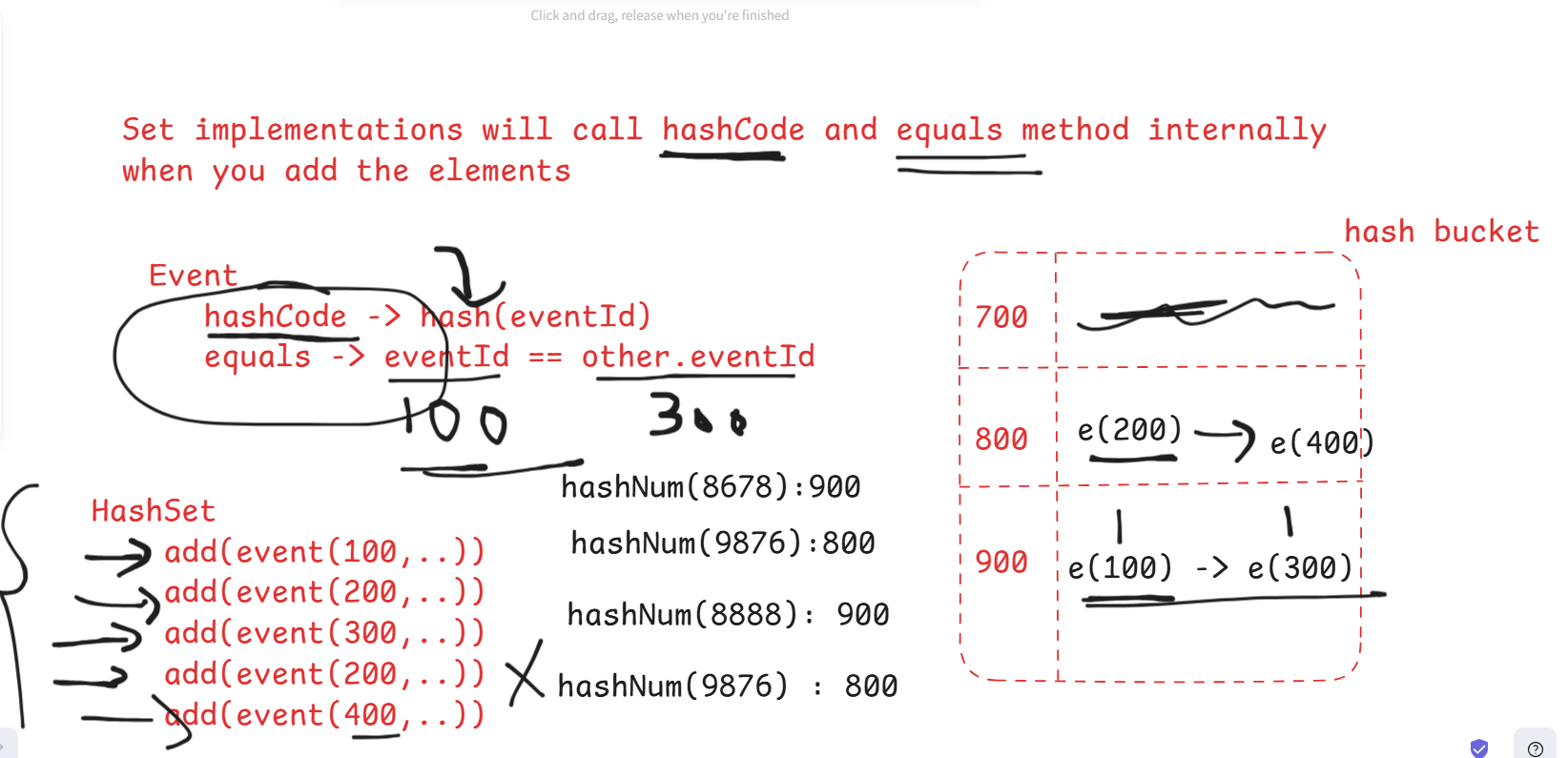


TestEvent.java



Whenever you add complex elements into the Set you must override 2 methods of Object class which are

1. equals(): comparing the objects whether they are equal or not
2. hashCode(): specifies where the object must be stored in the hash bucket



Sorting the event objects

You can add event objects into the TreeSet to keep in the sorting order, however TreeSet looks for Comparable or Comparator type to sort the objects, if they are not used then TreeSet throws ClassCastException.

Note: Simple types, String, LocalDate and few other classes are of Comparable type hence TreeSet can sort them, however custom classes not implementing Comparable can’t be sorted

How TreeSet can arrange the even objects using compareTo method.  
555, 666, 777, 888

event(777)  
event(666)  
event(888)  
event(555)  
event(444)

777-> compareTo -> compare(x, y) -> 0 -> 777(pivot)  
666 -> compareTo-> compare(666, 777) -> -1  
888 -> compareTo-> compare(888, 777) -> 1  
555-> compareTo-> compare(555, 777|666) -> -1

Comparable can be used for natural sorting, i.e., like a default sorting for complex type, you can use sorting id’s in ascending or as the default sorting technique, however when you need other properties to be sorted like name, startDate, endDate then you must use Comparator interface, which needs to be implemented outside the class that needs to sorted.

Comparator has one abstract method int compare(x, y)

You can implement Comparator using a class or nested class or anonymous inner class

Comparator<T> c1 = new Comparator<T>() {   
 public int compare(x, y) { return … }  
}  
You must pass this comparator object to the TreeSet(Comparator)  
Set<T> set = new TreeSet<T>(c1); // TreeSet uses Comparator to compare the object  
Set<T> set = new TreeSet<T>(); // TreeSet uses Comparable to compare the object