Core Java

Software Requirement

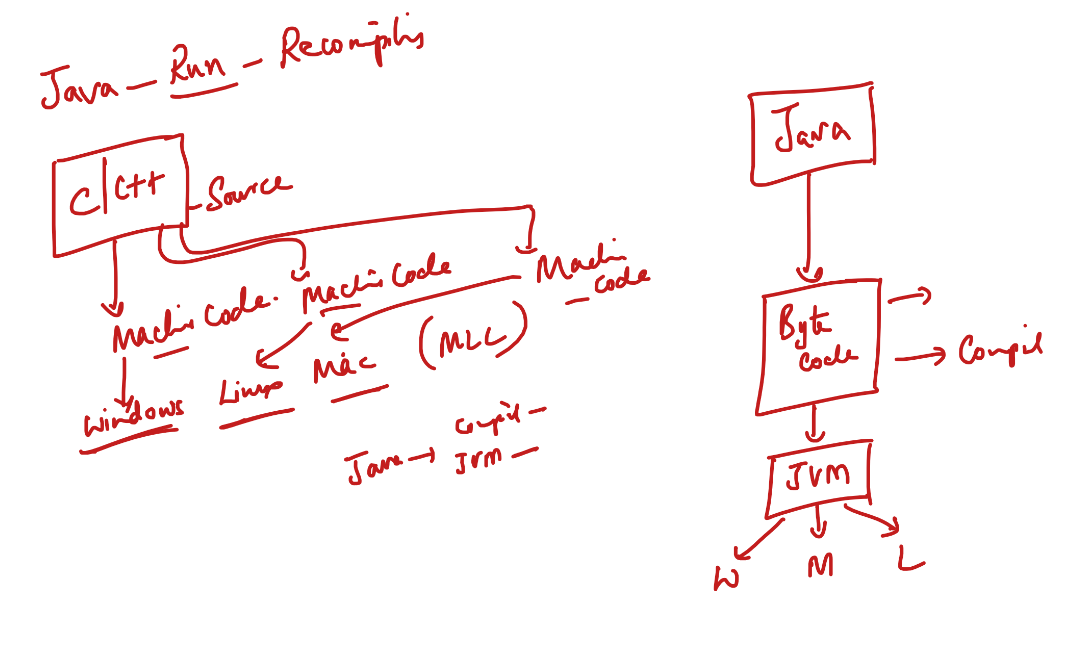
* JDK 17 or later
* Eclipse IDE / VS Code

How to verify java installation

Two commands

1. javac
2. java

Java: It is a Platform Independent & Object oriented programming language



Object Oriented Programming Language

It helps programmers to create real world entities in the application and make these real world entities to interact with another real world entities to complete any task.

Building blocks of any object oriented programming language

1. class: Blueprint of an object, that represents object structure
2. object: It is an instance of a class / real world entity that you can create from the class

Terms in Java

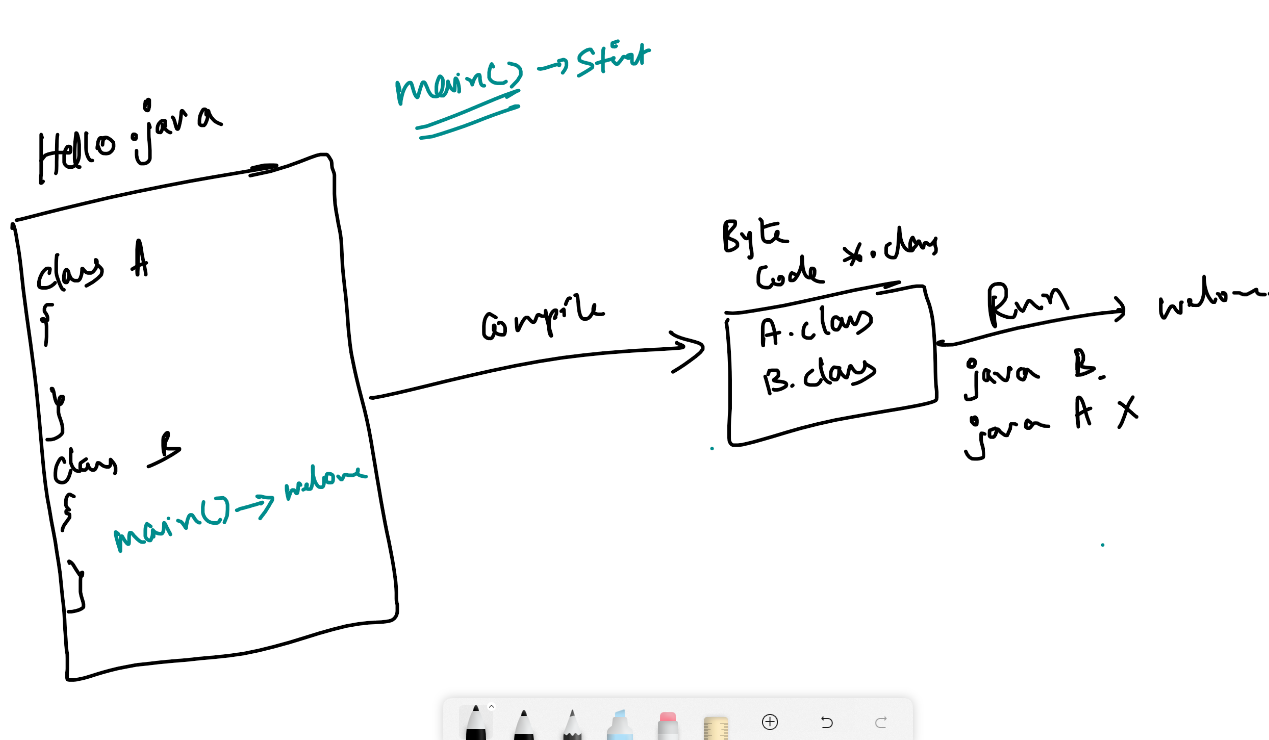
1. JDK: Java Development Kit – software that gives compiler and JRE both
2. JRE: Java Runtime Environment – place where your java application runs, it will have JVM and other components to load the application
3. JVM: Java Virtual Machine – an interpreter to convert byte code to machine code

Java versions: Java 8, Java 11, Java 17, Java 21

Java Editions:

1. JSE: Java Standard Edition – where you can create java programs in console based application
2. JEE : Java Enterprise Edition – where you can create java programs in web based applications

How to write our first java program



Fundamentals

1. Datatypes
2. Operators
3. Conditional Statements
4. Loops
5. Arrays
6. Classes & Objects

Datatypes: These are keywords used to create variables that specifies what kind of value it stores

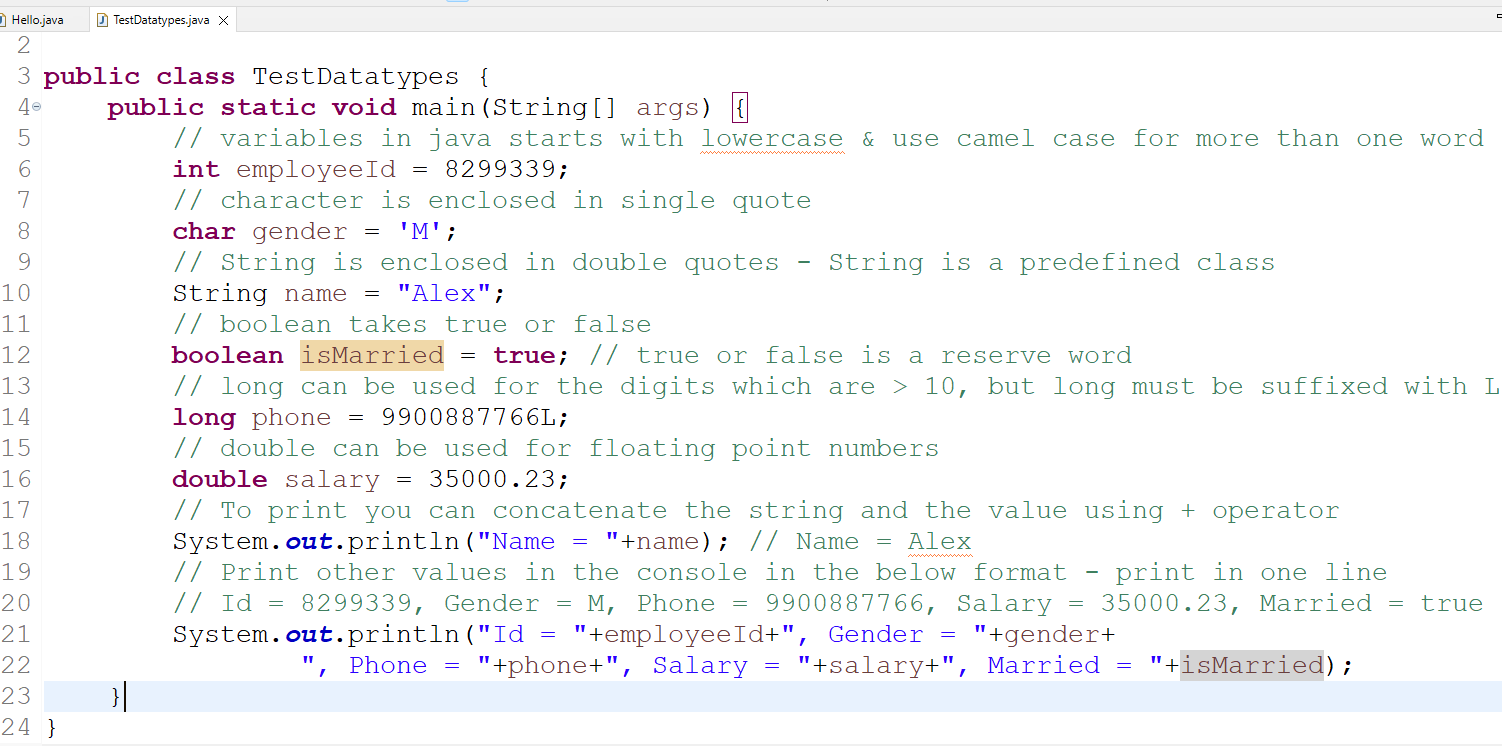
int x = 20;

double y = 55.4;

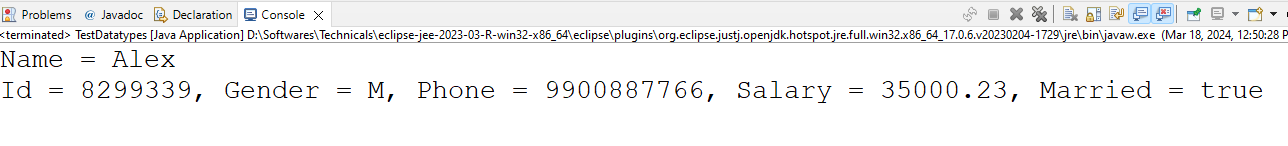
Java has following datatypes & each datatypes has fixed size

|  |  |  |
| --- | --- | --- |
| Datatype | Size in bytes | Range |
| byte | 1 | -128 to +127, including 0 |
| short | 2 | -32768 to +32767, with 0 |
| int | 4 |  |
| long | 8 |  |
| float | 4 |  |
| double | 8 |  |
| char | 2 |  |
| boolean | 1 |  |

TestDatatypes.java



Output:



Same program which takes input from the keyboard, instead of hard coding the values

In Java we have an inbuilt class Scanner in java.util package, we must import this

import java.util.Scanner;

Then we must create a Scanner object to take input from the keyboard

Scanner scan = new Scanner(System.in);

You must call inbuilt methods of Scanner to take different types of value

int x = scan.nextInt(); // to read int value  
double y = scan.nextDouble(); // to read double value  
String z = scan.next(); // to read a single string word  
boolean b = scan.nextBoolean(); // to read a boolean  
long l = scan.nextLong(); // to read a long

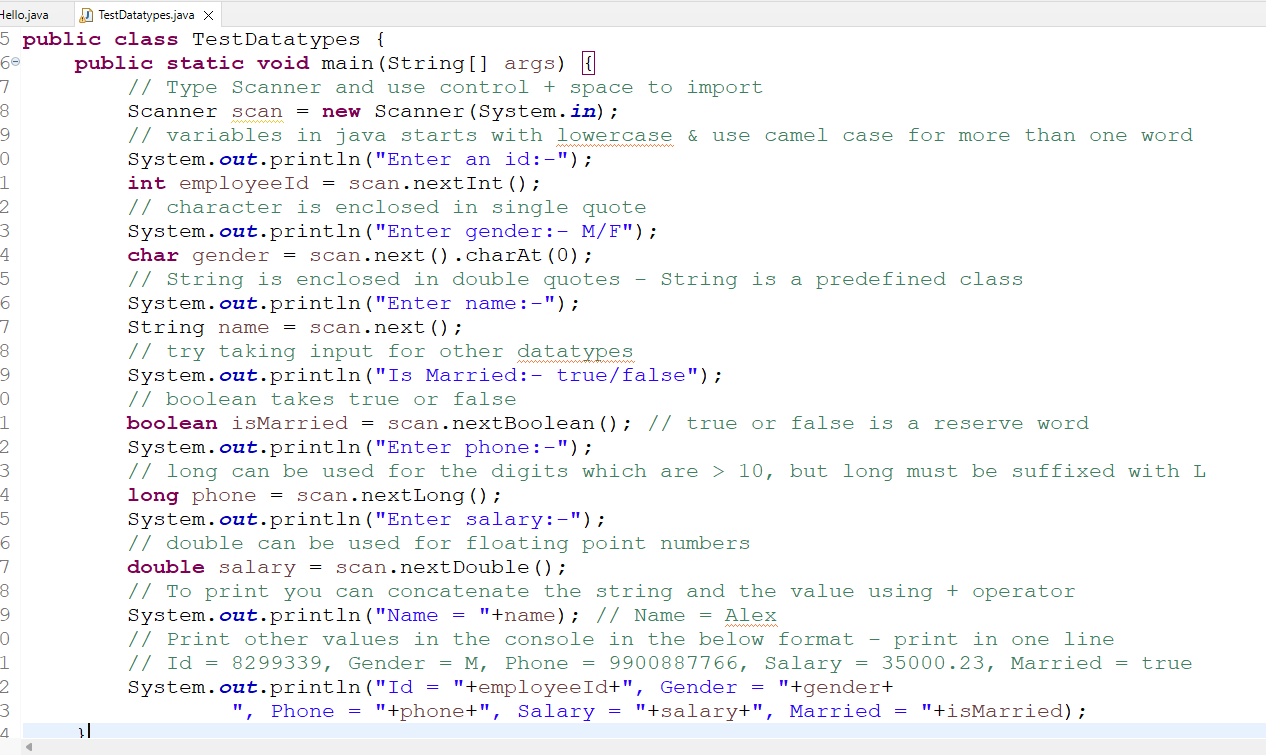
To read character you don’t have a method in scanner, you must use String and call the charAt(index) method to get a character of a specific string value

char ch = “Hello”.charAt(1); // ch = e

String z = scan.next();  
char ch = z.charAt(0); // very first character

char ch = scan.next().charAt(0); // ch = very first character of the string

TestDatatypes.java



Operators:

=, +, -, \*, /, <, >, <=, >=, ==, ++, --

x = 5;

y = x++; // post increment

Conditional Statements

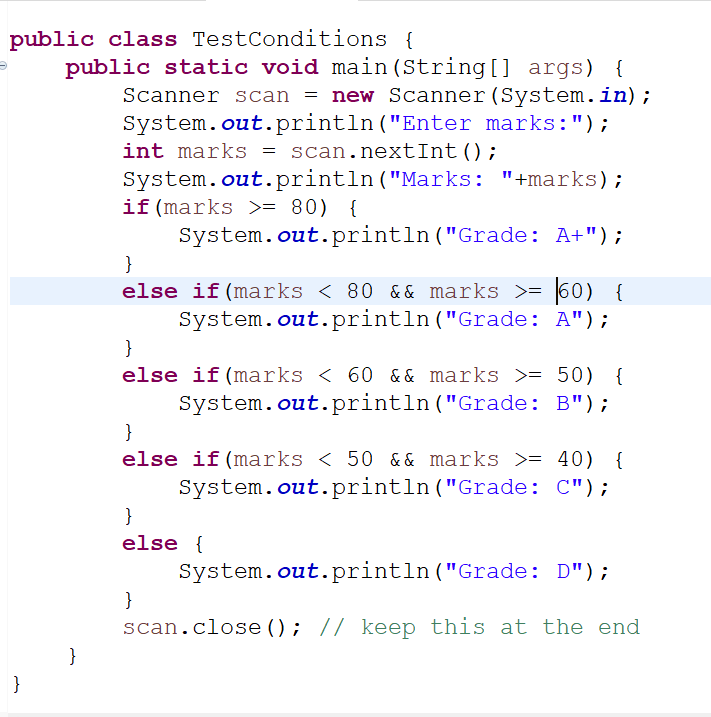
if

if else

if else if else if else

switch

TestConditions.java



TestConditions.java with switch

switch(options) {  
 case optionValue: statements;  
 break;  
 case optionValue: statements;  
 break;  
 default : statements  
}

Here optionValues can be int, String, char  
case 1 or case “someString” or case ‘A’

Loops

1. for
2. while
3. do while

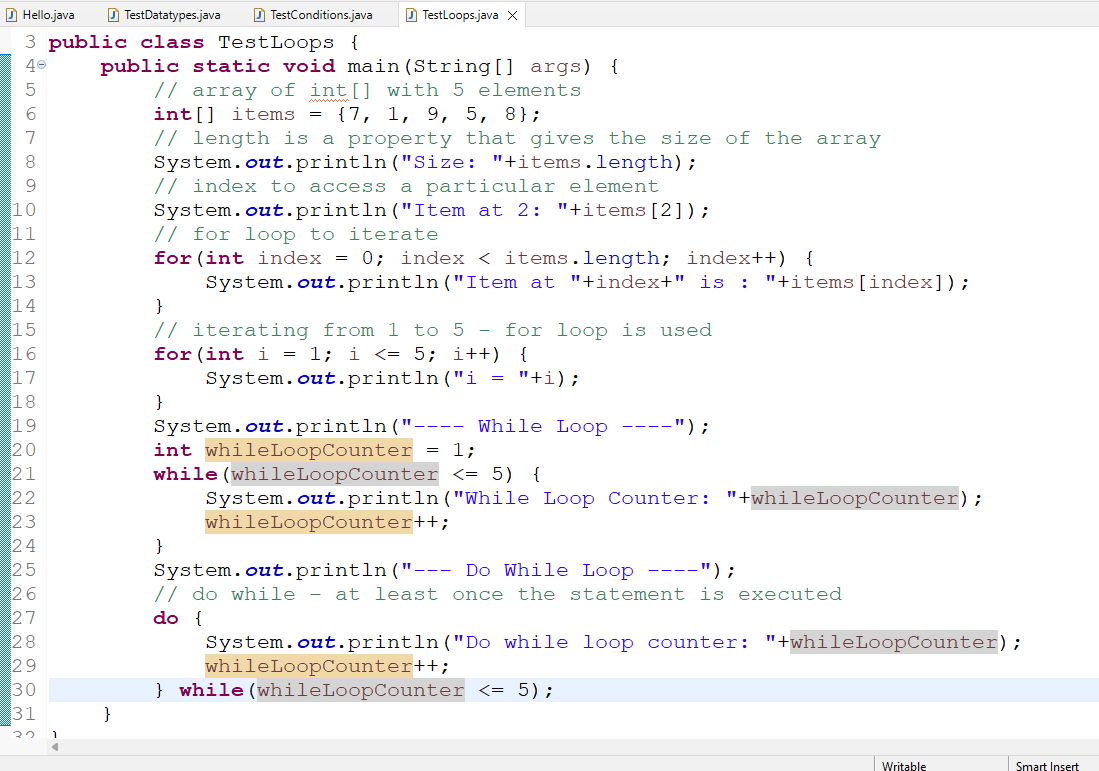
for loop: When you know how many times you want to iterate

while & do while: When you don’t know how many times you want to iterate but you want to iterate until some condition true

while: first condition is checked & then executed, its like checking the condition in the beginning

do while: At least once the statement executed and then the condition is checked, its like checking the condition while exiting

TestLoops.java



Activity:

1. Create an array of numbers and print the maximum and minimum value in the array & the sum of all the elements in the array using only one for loop

ex: items = {7, 1, -7, 8, 9, 3}, Max = 9, Min = -7, Sum = 21

1. Enter a 3 digit number and print each digit in words

ex: Input = 762, Output = Seven, Six, Two

1. Enter a 3 digit number & add the highest and lowest digit of the 3 digits

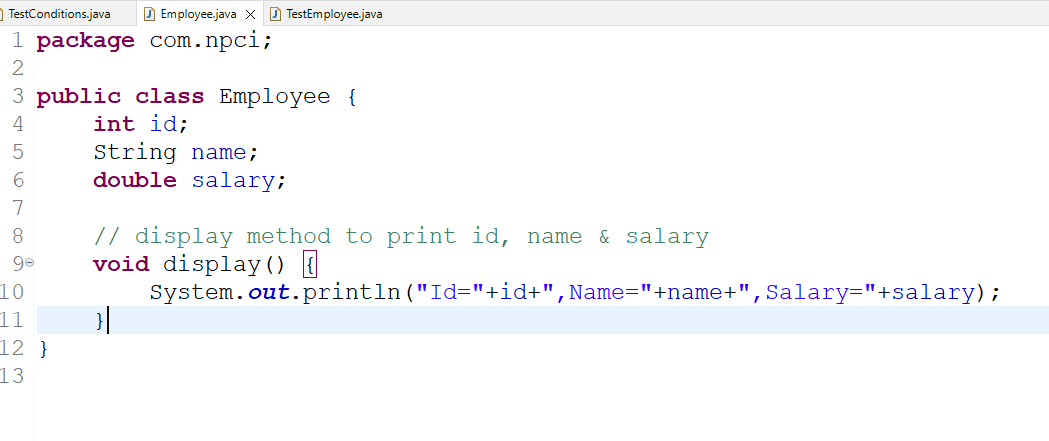
ex: Input = 762, Output = 7 + 2 = 9

Classes & Objects

class can have three things mainly

1. variables/fields/properties: Stores data
2. methods/functions: Performs action, it will have logics
3. constructors: Its like a methods but its sole job is to initialize the properties

Employee.java



TestEmployee.java



Constructors: These are called when object is created, it will be created by default if the class doesn’t have any constructor, such constructors are called as default constructors which doesn’t take any parameter, however you can overload the constructor by passing parameters

class Employee {   
 Employee(int id, String name, double salary) { //parameterized constructor }  
}

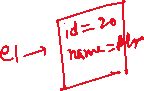
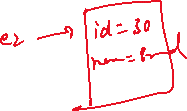
class Employee has only one constructor, default constructor wouldn’t be created.

class Employee {   
 Employee() { //default constructor }  
 Employee(int id, String name, double salary) { //parameterized constructor }  
}

Now class Employee has 2 constructors

Parameterized constructor can be used to initialize the properties

class Employee {   
 int id;  
 String name;  
 Employee(int id, String name) {   
 this.id = id;  
 this.name = name;  
 }  
}



Employee e1 = new Employee(20, “Alex”);  
Employee e2 = new Employee(30, “Brad”);

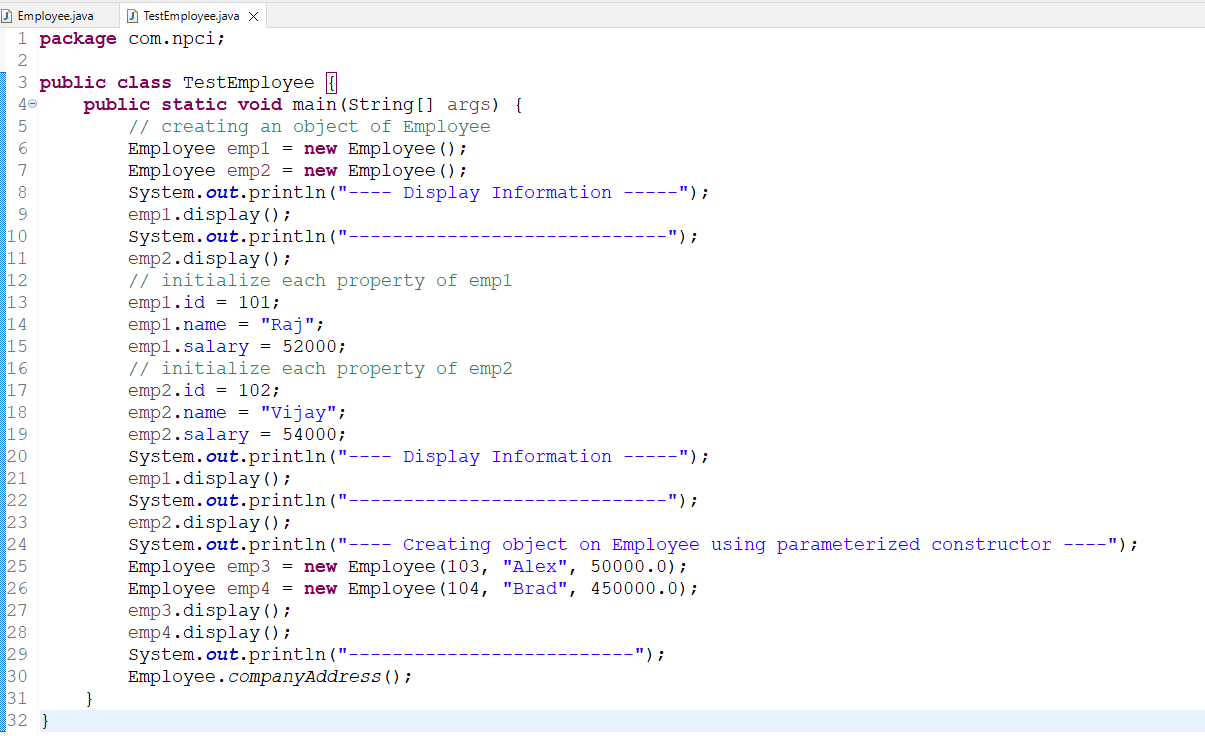
Static members:

These are the members which you can access without creating object, i.e., using the class name, you can create static variable & static methods in a class that you can call directly using the class name.

Employee.java



TestEmployee.java



Note:

1. Static methods cannot directly access non-static members(variables or methods) of the same class, it must be accessed using the object reference
2. Non-static methods can directly access static members of the same class

Final: Can be used on variables, methods & classes

final variables are constants you can’t modify

i.e., final int x = 20; // you can’t change the value of x any more

final methods & final classes we will discuss after the inheritance

class Employee {   
 final static String companyName = “NPCI”; // constant  
 static int employeeCounter = 0; // not a constant you can modify  
 Employee(…) {  
 companyName = “NPCI”; // compile time error  
 employeeCounter++;  
 }  
}

OOPs principles

1. Encapsulation
2. Inheritance
3. Polymorphism
4. Abstraction

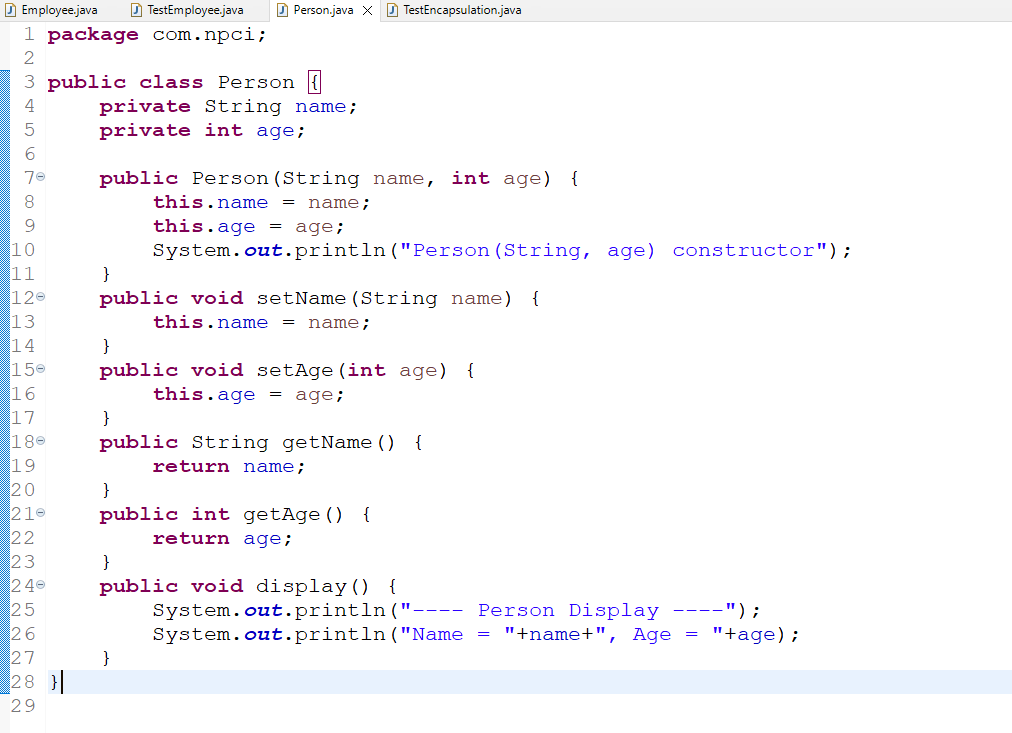
Encapsulation: It is used to hide the data so that it can’t be accessed directly outside the class, i.e., the data is restricted from the direct access to avoid code to give some invalid value

In Encapsulation you need to access this data using public setters & getters methods

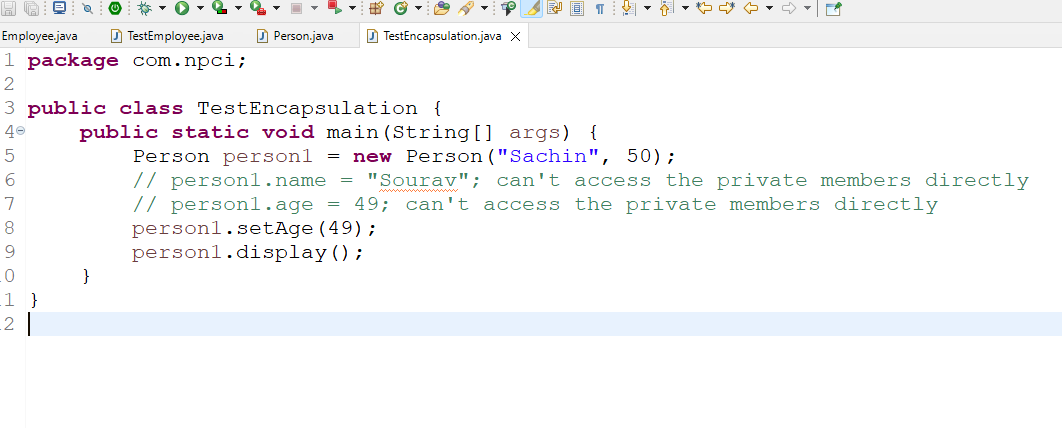
setter method: This modifies the data

getter method: This returns the data

Person.java



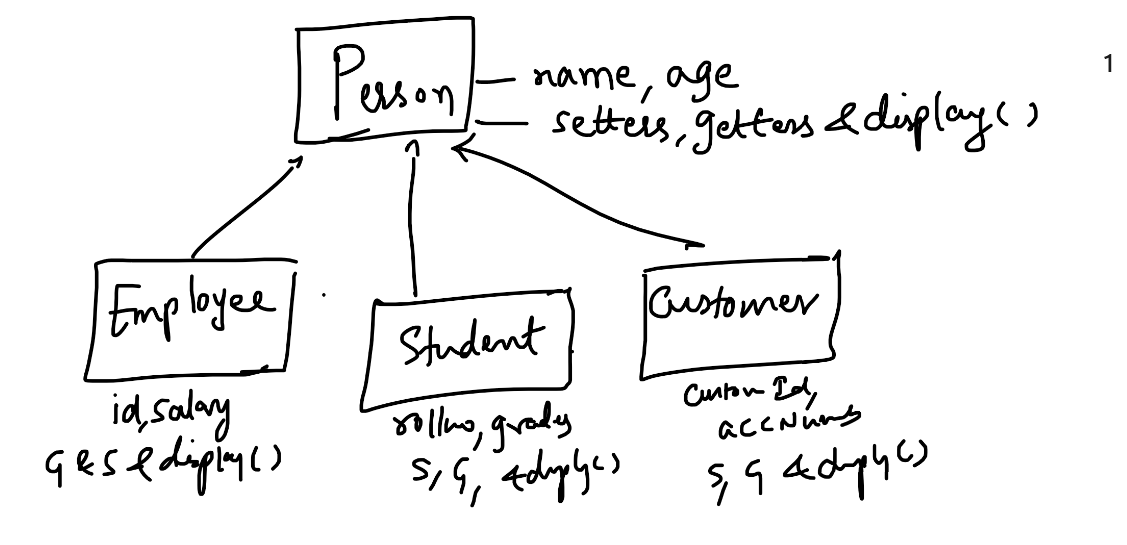
TestEncapculation.java



Note: In this example we have made the properties private so that outside the class they can’t be accessed directly, the only way is to access them using getters & setters

Inheritance

It acquires common properties & behaviors of a class from another class



Note:

1. Every constructor of the subclass by default calls the parent class default constructor
2. super() is the statement used to call the constructor, this must be present only in the first line of the constructor, keeping anywhere else leads to an error.

Object: It is the root class for all the classes, it has some common features which will be inherited to every class

Activity:

Create a Student class with rollNo, name, age and array of marks properties, initialize the name & age from the super class & override display method to print student properties, display method must calculate average marks and print the average marks

ex: if marks array has 6 marks then display prints the average value of 6 marks

In the main class create a Student object by passing rollNo, name, age and array of marks having 6 subjects & call the display method

Polymorphism: A method that has multiple forms or can give multiple results

1. method overloading: same method names but different signature, compiler can predict which method will be called in method overloading
2. method overriding: same method & same signature, but different logics in the subclass, only at runtime the method call can be predicted, here a single statement can give multiple results

System.out.println(2);

System.out.println(“Hello World”);

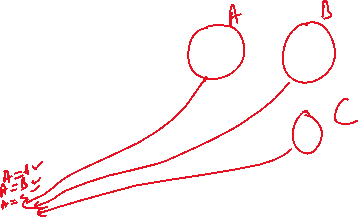
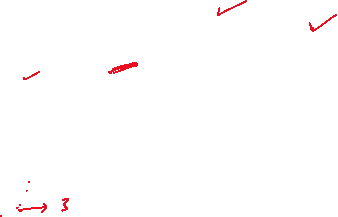
System.out.println(x); // println can take all types of data like boolean, int, long, String, Object

class Calculator {  
 void add(int x, int y) { }   
 void add(double x, double y) { }   
}

add(20, 30); // add(int, int)  
add(20.3, 40.3); // add(double, double)

Method overriding

class A {   
 m1() { … }   
}  
class B extends A {   
 m1() { … }  
}  
class C extends B {   
 m1() { … }  
}



void test(A x) {   
 x.m1();  
}



Abstraction

Hiding the complexity and showing only the necessary details to the end user, this makes code loosely coupled

It is achieved in 2 ways

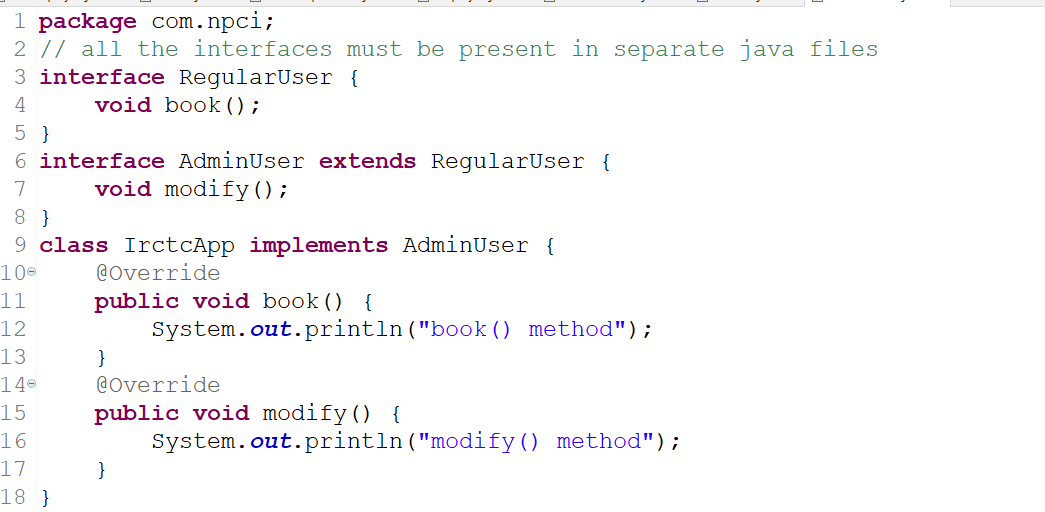
1. interface
2. abstract class

interface: It is a contract that specifies what methods do instead how it is implemented, it will have only method declarations (methods without logic)

interface X {   
 void m1(); // public abstract void m1();  
 void m2(); // public abstract void m2()  
}

class One implements X {   
 // class must implement all the abstract methods of the interface  
}  
class Two implements X {   
 // class must implement all the abstract methods of the interface  
}

TestInterfaces.java

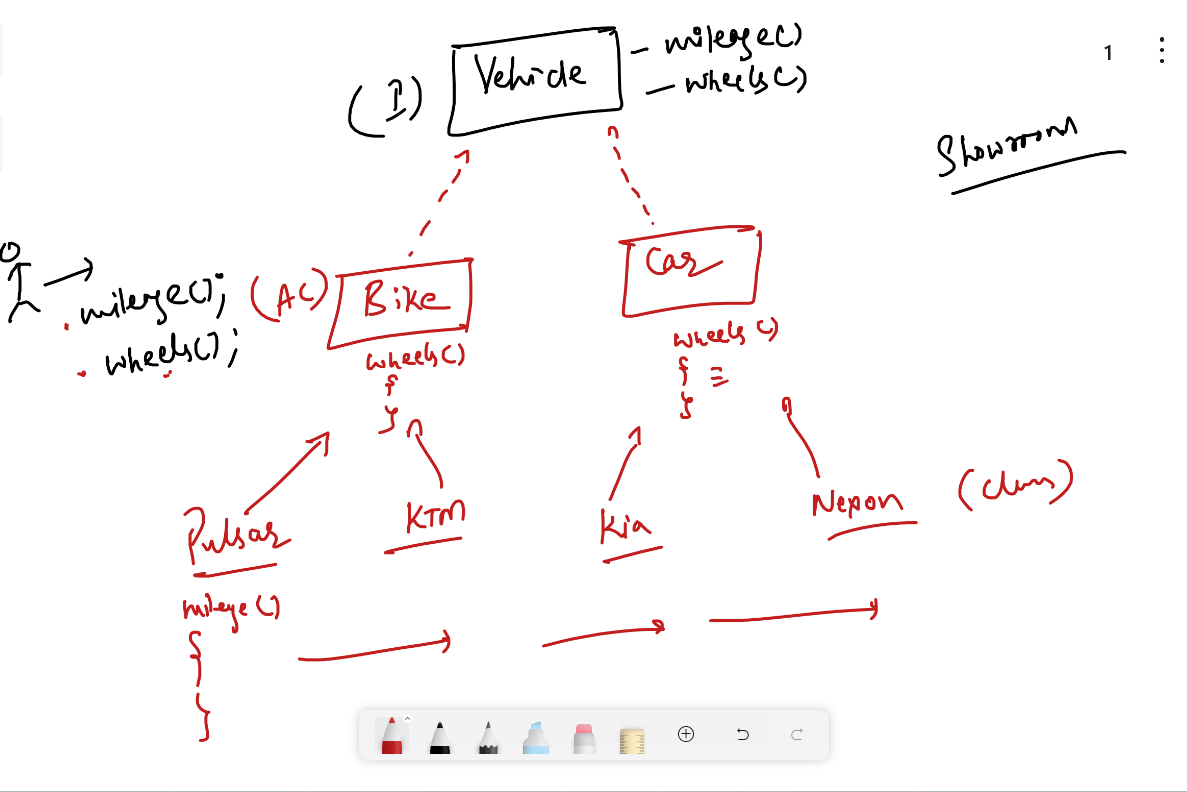
Important Note on interface

1. You can’t create object of interfaces
2. You can’t create constructors in interface
3. Methods are by default abstract
4. Variables are by default static & final i.e., constants

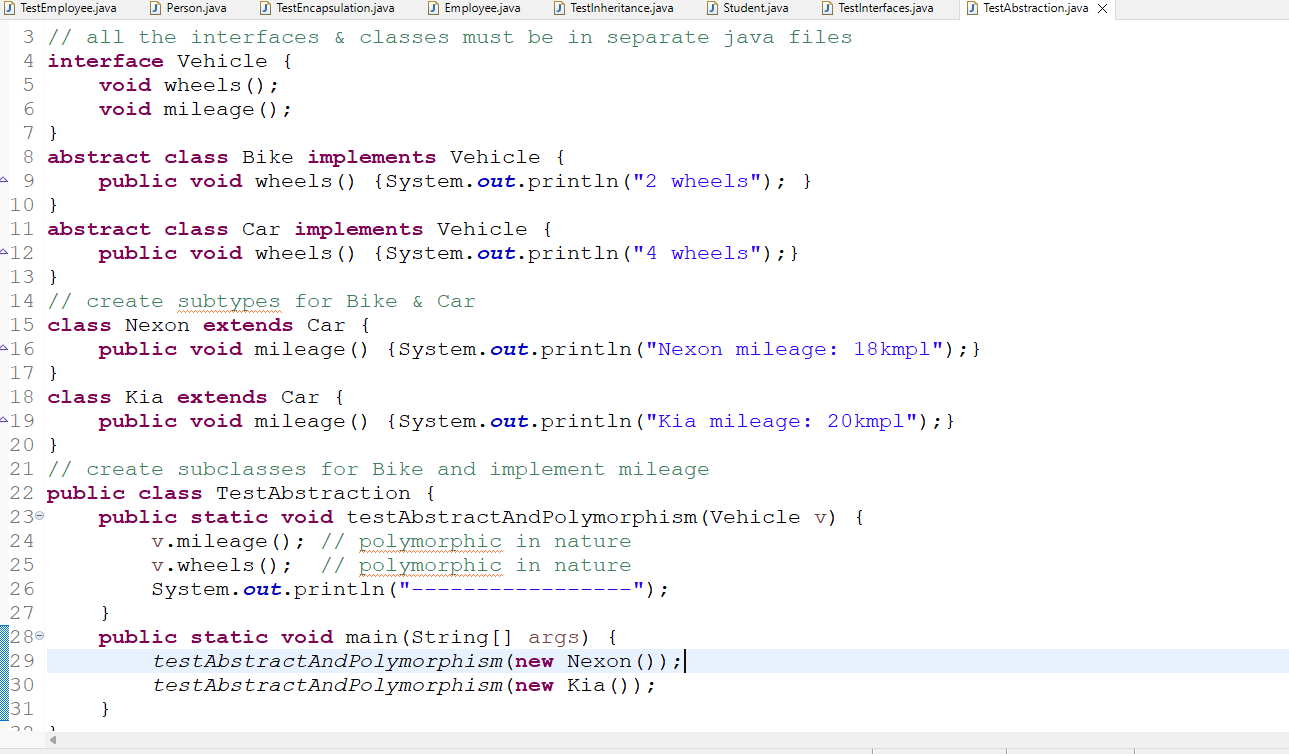
abstract class: It can have abstract methods & methods with body both, it can also have constructors, but you can’t create object of abstract class

abstract class Employee {   
 id, name, salary;  
 Employee(id, name, salary) { …. }  
 void display() { print id, name & salary }  
 abstract void incrementSalary();   
}

class Manager extends Employee {   
 // implement abstract methods – raiseSalary() – 25%  
}  
class Developer extends Employee {   
 // implement abstract methods – raiseSalary() – 20%  
}



TestAbstraction.java



Exception Handling

Exceptions are runtime errors which can stop the application from running if not handled, there are mainly 5 keywords

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

try: You write the codes which might generate the exception

catch: You write the codes to handle the exception

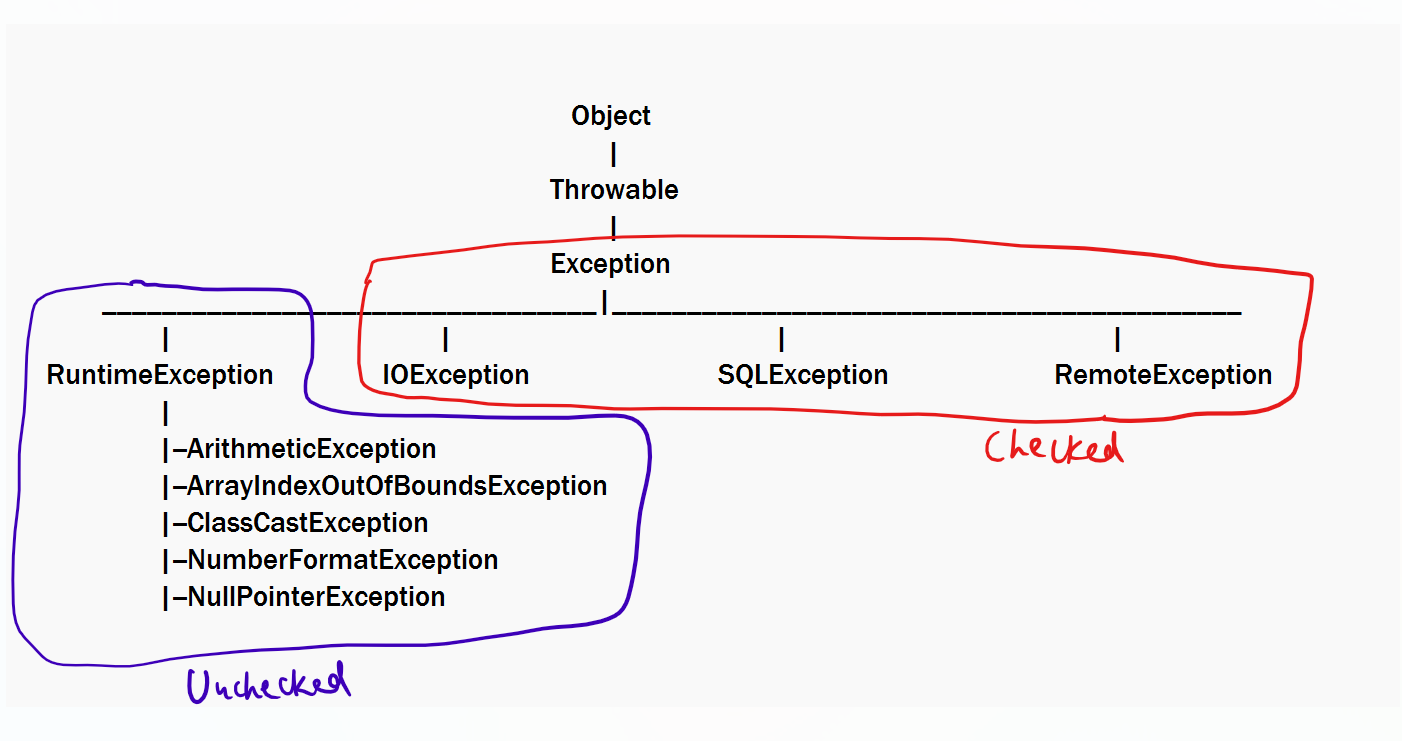
finally (optional block): You write the codes that needs to be mandatorily executed

throw: It is used to manually throw the exception or to manually generate the exception in the code

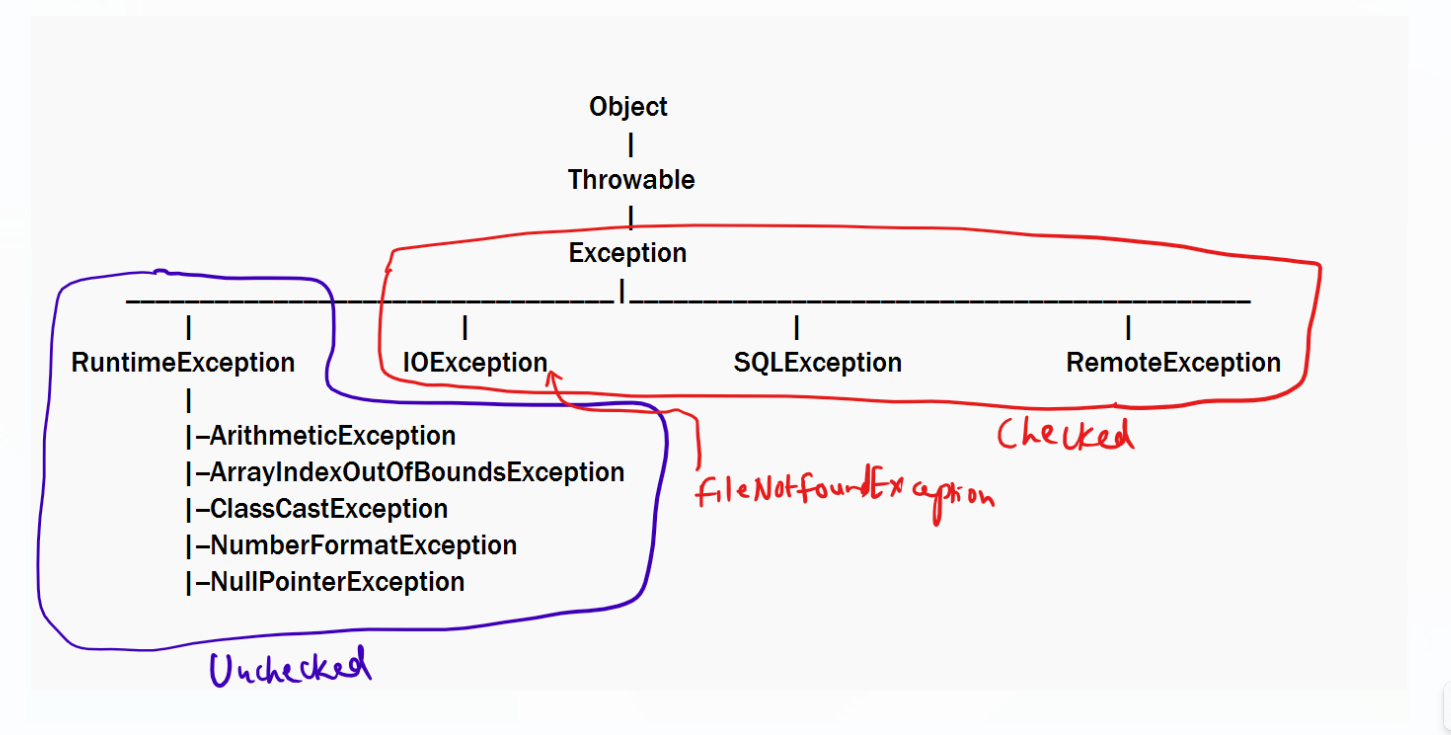
throw new ExceptionName();

throws: It is used to propagate the exceptions to the caller, when the caller knows how to handle the exception using try/catch

Exception Hierarchy



IOException has a subclass FileNotFoundException



Checked Exceptions: These needs to be handled at the compilation time itself, because they are checked at the compilation time – You must either use try/catch or throws

Unchecked Exceptions: These are not checked at the compilation time, they are optional to handle, however if the unchecked exception occurs & not handled then the program terminates

Custom Exception: These are the exceptions that are user defined, you can create custom exception by inheriting any one of the exception class

1. extend Exception to create custom checked exception
2. extend RuntimeException to create custom unchecked exception

File Handling: Reading & Writing operations

We have some inbuilt classes like FileReader, FileWriter which have methods that throws checked exceptions like IOException, FileNotFoundException

FileReader: it is used to read text files, it has following constructors & methods that throws checked exception

1. FileReader(String path) throws FileNotFoundException
2. read() throws IOException

FileWriter: It is used to write text files, it has following methods that throws checked exception

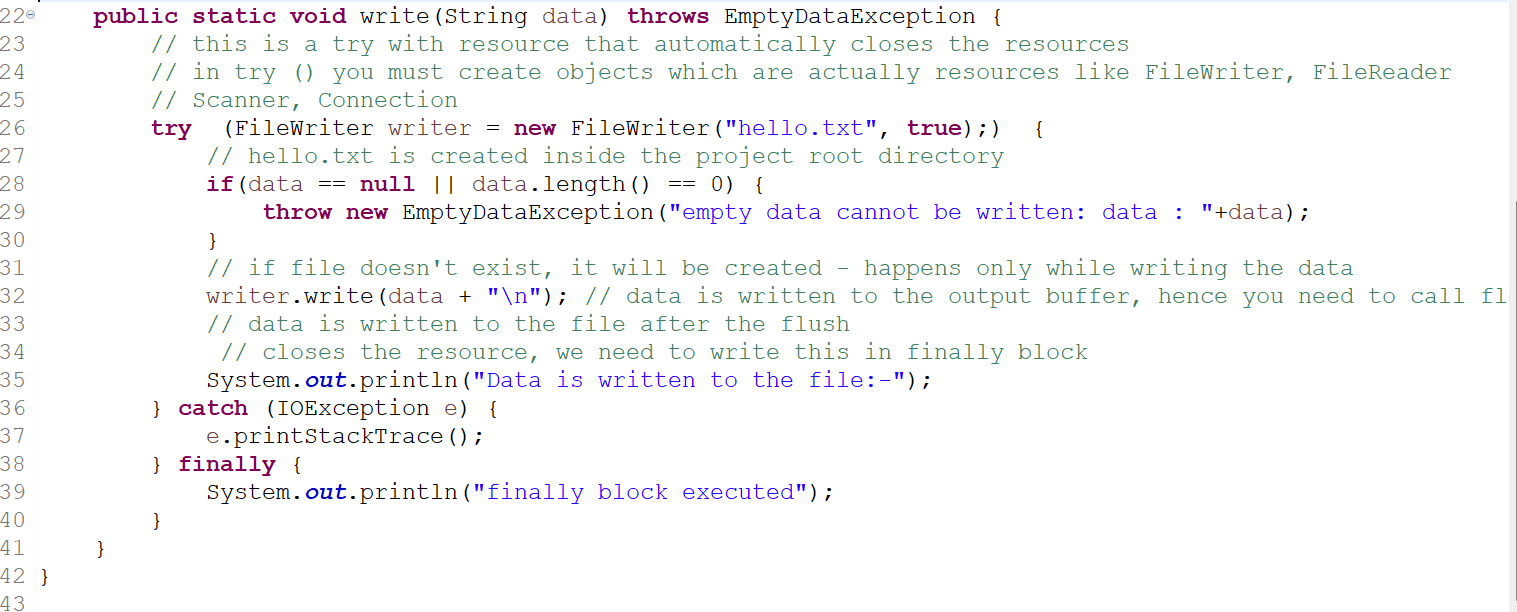
1. FileWriter(String path) throws IOException
2. write(char c) throws IOException

Handling all the exceptions using super class

try {   
 // AIOBE  
 // AE  
 // IOE  
 // SQLE  
} catch(IOE e) {}   
catch(Exception e) { }

try with resource closing syntax

TestExceptions.java



Above code uses try – with resource syntax, which automatically closes the resources like Scanner, FileWriter, FileReader and etc, it also does auto-flush hence you don’t have to use flush() method as well

Inbuilt classes: There are lot of packages in java like

java.lang, java.util, java.io and so on

classes of java.lang you can use without importing

java.lang classes

Object, String, System, Thread, RuntimeException, Exception,

Object:

It is a root class in java, all the classes extend Object class so that common features every class must have will be inherited like

* toString(): Represent the object in string form when you print an object, by default it returns memory address in hexadecimal format
* equals(): Compares two objects memory address by default
* hashCode(): Returns an unique number of an object based on the address

Employee employee = new Employee(101, “Raj”, 53500);  
System.out.println(employee); // it calls employee.toString() internally

LocalDate date = LocalDate.now(); // inbuilt class from java.time

ISO format : yyyy-MM-dd

LocalDate.now() creates an object of LocalDate of current date

System.out.println(date); // it calls date.toString() internally – 2024-03-20

LocalDate date2 = LocalDate.of(1970, 1, 25);

System.out.println(date2); // 1970-01-25

LocalDate date3 = LocalDate.parse(“1971-01-26”);

System.out.println(date3); // 1971-01-26

String s = “hello”;

System.out.println(s); // s.toString() internally

catch(Exception e) {   
 System.out.println(e); // e.toString() internally // java.lang.Exception  
}

String: It creates immutable string object, once the string is created you can’t modify

String uses two memory pools to store the string objects

1. String constant pool: It will have only one copy of a particular string
2. String non-constant pool: It can have multiple copies of a same string

s1 = “hello”;

s1.concat(“123”);

System.out.println(s1); // hello

s1 = s1.concat(“123”);

System.out.println(s1); // hello123

s2 = “hello”;

s3 = new String(“hello”);

s4 = “hello”;

s2.equals(s3); // true

s2 == s3; // false

s2 == s4; // true

s1 = “icici”;

Note: In String equals() method is overridden from the Object class to compare the content, by default Object equals() method compares memory address

String Buffer: It creates mutable string object, its content will be modified

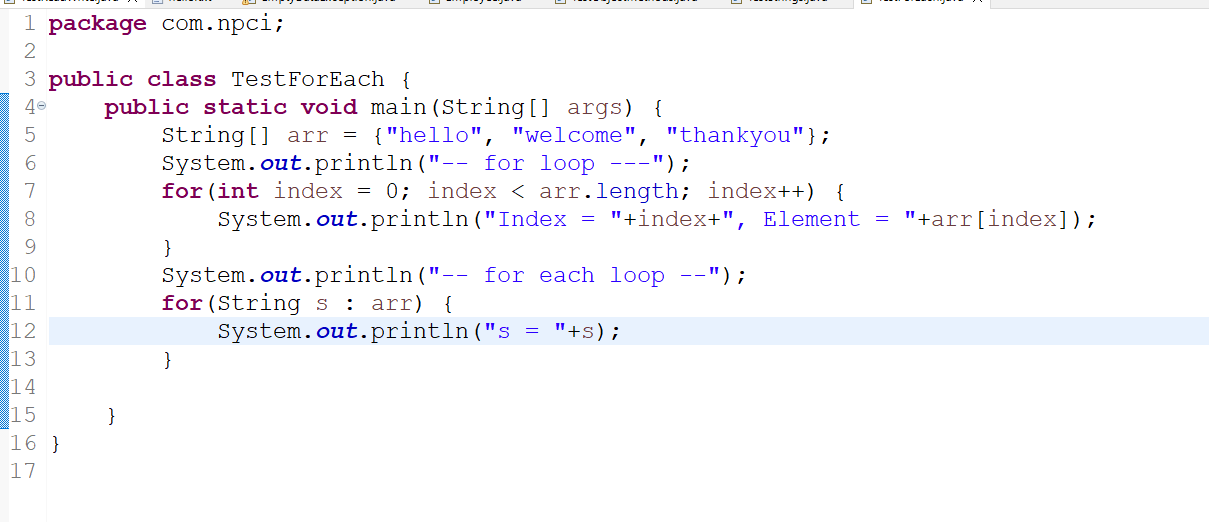
StringBuffer has overridden toString() method to return the string, however equals() method of Object class is not overridden

StringBuffer sb1 = new StringBuffer(“hello”);  
StringBuffer sb2 = new StringBuffer(“hello”);

System.out.println(sb1); // hello  
sb1.equals(sb2); // false

sb1.append(“123”);  
System.out.println(sb1); // hello123

Enhanced For Loop: simplifies iterating over array / collections, it doesn’t use any index or compares with any value to iterate

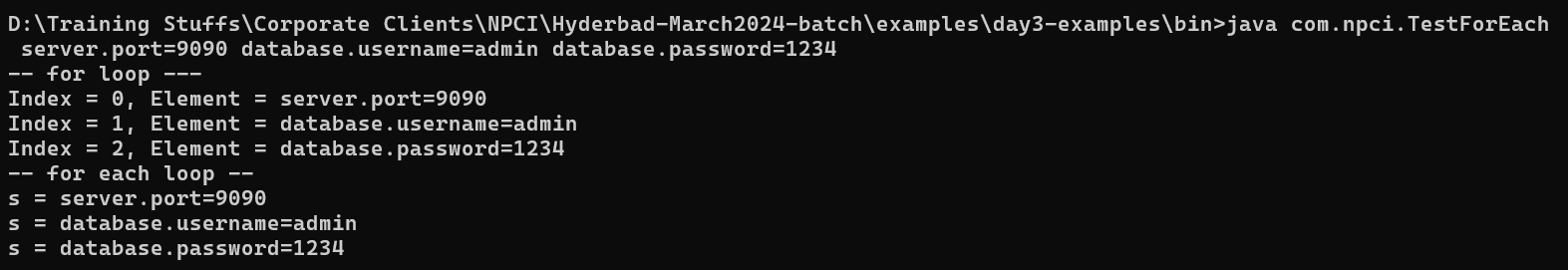


Command line argument: main(String[] args) is a command line argument which you can pass at runtime before executing the program

java Test arg1 arg2 arg3 arg4

main method String[] stores all the arguments which you can iterate

java Test server.port=9090 database.username=root database.password=1234



Multithreading

A single program or an application that can perform multiple tasks at the same time.

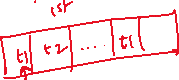
In Java we have Thread & Runnable to write multi-threaded applications

1. Runnable: It is an interface which has an abstract method: run()

run() is an entry point for the threads

1. Thread: It is a class which is used to create thread object & manage the threads like starting, stopping

class X implements Runnable {   
 public void run() { …. }  
}



X x1 = new X();



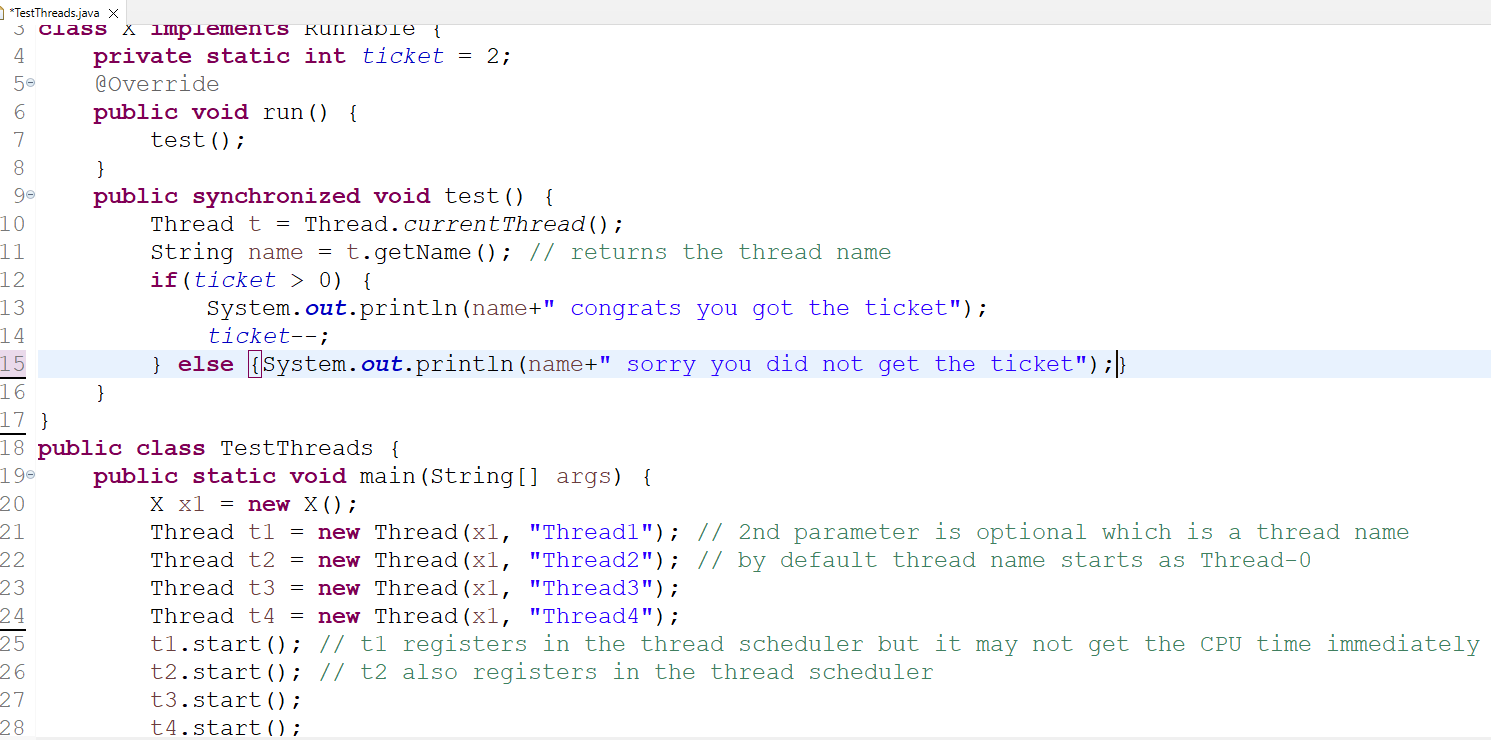
Thread t1 = new Thread( x1 );  
Thread t2 = new Thread ( x1 );



t1.start();  
t2.start();



Thread synchronization: It is a process of making only one thread to run the method / block of code without allowing other threads until the thread completes



Memory management in Java

Stack: Loads classes, static variables, static methods, local variables

Heap: Stores only object (these objects are removed by Garbage collector)

Reference:

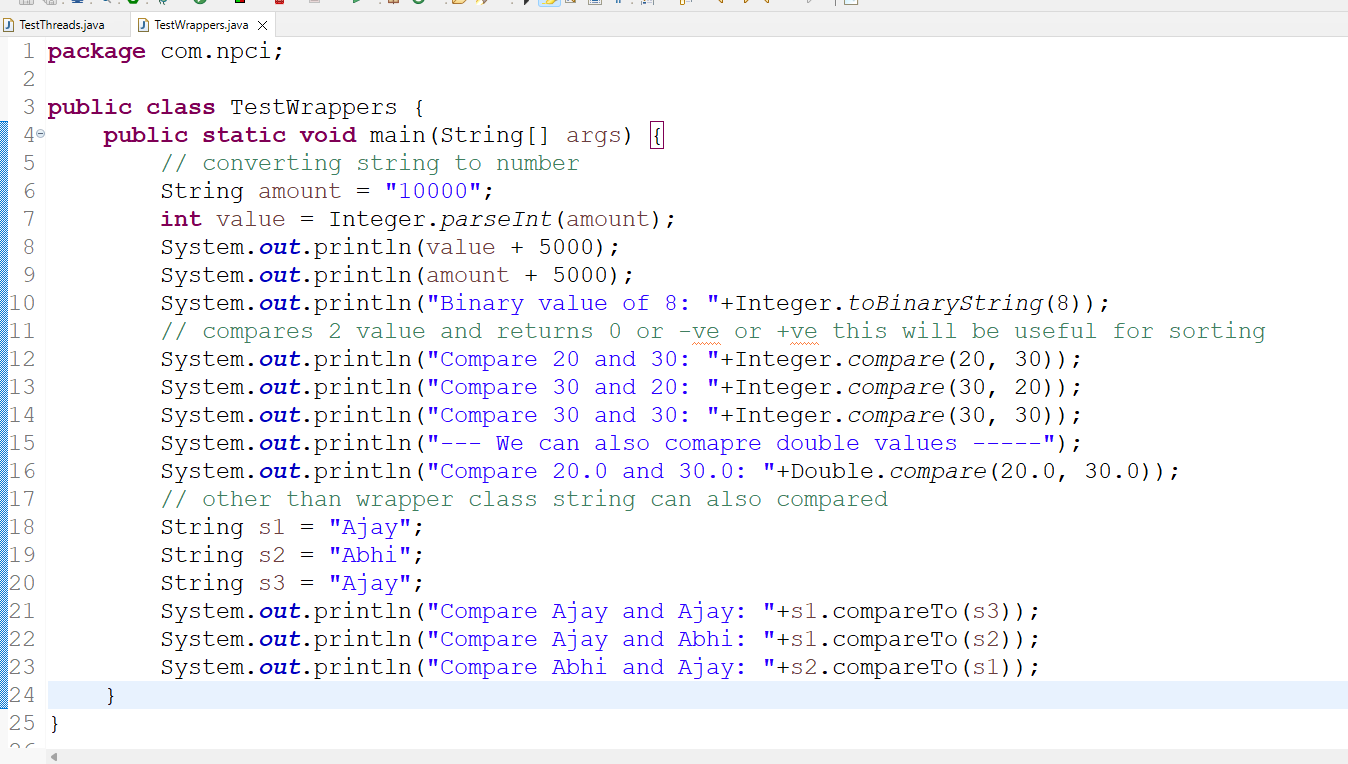
<https://newrelic.com/blog/best-practices/java-garbage-collection#:~:text=Garbage%20collection%20in%20Java%20is,be%20executed%20by%20a%20JVM>.

Wrapper classes

These are the classes defined for the primitive datatypes like

1. byte – Byte
2. short – Short
3. int – Integer
4. long – Long
5. char – Character
6. double – Double
7. boolean – Boolean
8. float – Float

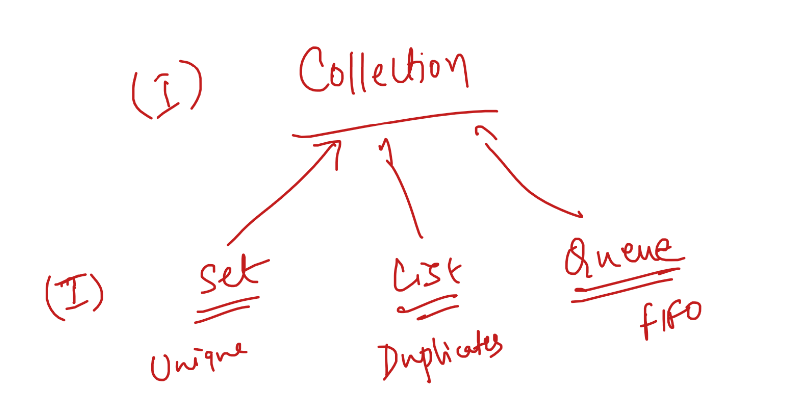
Primitives can perform arithmetic operations & wrapper classes perform arithmetic operations & other operations with the help of predefined methods



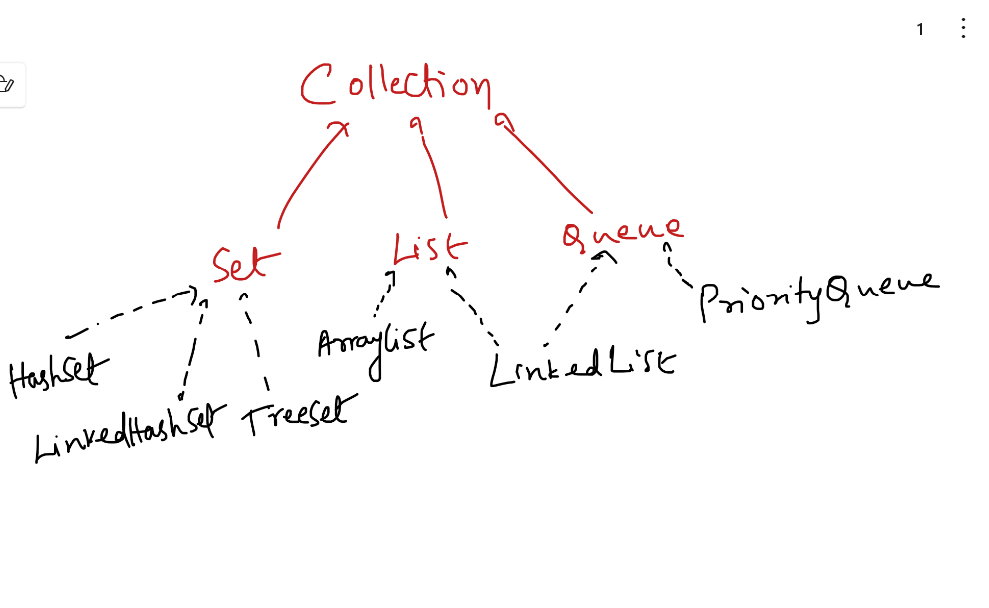
Collection Framework – Datastructure to maintain the data

Collection is an interface that provides methods like

* add(Object)
* remove(Object)
* clear()
* iterator()
* size()



All these interfaces are implemented in java already

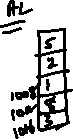
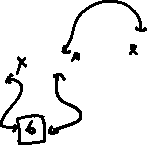
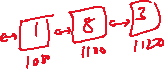


Set has 3 implementations, all of them allow no duplicates

1. HashSet: Random order, but its retrieval is faster because it uses hash algorithm
2. LinkedHashSet: Maintains the data in insertion order
3. TreeSet: Maintains the data in sorted order

List has 2 implementation, it is index based & also support duplicates

1. ArrayList: It maintains the elements in contiguous memory address
2. LinkedList: It maintains the elements in non-contiguous memory address



Queue: It is mainly used to process the data after removing, it also supports duplicates, it is implemented by

1. LinkedList: acts like FIFO – queue provides methods to remove in FIFO
2. PriorityQueue: queue provides methods to remove in sorted order

Map: It is also a datastructure, but it is not part of collection hierarchy, it maintains the data in key & value pairs, here key must be unique, Map is an interface has following methods

* put(key, value);
* get(key)
* remove(key)

Map has 3 implementations just similar to Set implementations

1. HashMap: random order
2. LinkedHashMap: insertion order
3. TreeMap: sorted order

Collection Examples

List<Employee> list = new ArrayList<Employee>(); // you can store only employee objects

List list = new ArrayList(); // you can store multiple types of data, employee, customer,

Note: Storing multiple types in a single container is not recommended

Set<Employee> set = new HashSet<Employee>() or   
Set<Employee> set = new HashSet<>();

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

// you can iterate using for loop

for(Employee e : list ) {  
 System.out.println(e.getId()+”, “+e.getName());  
} // for loop can only iterate it cannot remove the elements while iterating

Iterator<Employee> it = list.iterator(); // it can iterate and also remove while iterating  
// Iterator has 3 methods: hasNext(), next(), remove()

while(it.hasNext()) {// hasNext checks if next element is present   
 Employee e = it.next(); // returns the iterated element  
 // suppose you want to remove the element whose id = 99  
 if(e.getId() == 99) { it.remove(); }  
}

Comparable: It provides a method to compare 2 elements, a class can implement this to provide a natural sorting, it has a method compareTo(x)

Comparator: It is to customize the sorting technique, it is implemented in a separate class not in the same class, it is also an interface with only one method i.e., compare(x, y)

TreeSet can call this compare(x, y) instead of compareTo(x) when you pass the Comparator implementation object in the constructor

Set<Employee> s = new TreeSet<>(); // uses compareTo of Employee  
Set<Employee> s2 = new TreeSet<>( comparatorImpl ); // compare(x, y) of comparatorImpl